Australian Government
Department of Defence
Defence Science and
Technology Organisation

Shape Optimisation of Holes in Loaded Plates by Minimisation of Multiple Stress Peaks

Witold Waldman and Manfred Heller

Aerospace Division
Defence Science and Technology Organisation

DSTO-RR-0412

ABSTRACT

A new method has been developed for simultaneously minimising the peak tangential stresses on multiple segments around the boundary of a hole in a uniaxially-loaded or biaxially-loaded plate. It is based upon iterative finite element analysis. The efficacy of this new multi-peak method is demonstrated by 2D and 3D numerical examples, some of which include significant geometric constraints. A comprehensive series of benchmarks is explored in some detail and sets of useful transferable coordinates of optimised hole shapes for selected load and geometry cases are provided. It is shown that, when the optimal shape is achieved, the separate tensile and compressive stress segments around the hole boundary converge to constant stress regions of different values. The optimal hole shapes produce significant reductions in peak stress for all regions around the hole boundary, as compared to typical non-optimal circular holes. As the most fatigue critical location in a structure may not necessarily be the one with the biggest active tensile peak, it is desirable to be able to minimise these other stress peaks around the hole boundary. Hence, optimal shapes computed using the multi-peak method are useful for critical structural regions where it is desirable to substantially increase or maximise the fatigue life.

Published by

Aerospace Division
DSTO Defence Science and Technology Organisation
506 Lorimer Street
Fishermans Bend, Victoria 3207, Australia

Telephone: 1300 333 362
Fax: (03) 9626 7999

AR-016-270
April 2015

APPROVED FOR PUBLIC RELEASE

Shape Optimisation of Holes in Loaded Plates by Minimisation of Multiple Stress Peaks

Executive Summary

Aerospace Division has developed many technologies that reduce the cost of ownership of aircraft in service with the Royal Australian Air Force (RAAF) by extending the fatigue lives of airframe structural components. This is usually achieved by developing repair designs that can be applied to areas where stress concentrations have caused a location to become fatigue critical. A shape optimisation technology, based on using iterative finite element analyses, has been developed and used by the Defence Science and Technology Organisation (DSTO) to determine optimal repair profiles with multiple minimised local peak stresses under operating conditions. Previous work has utilised single-peak shape optimisation methods, which work only on the dominant peak. However, many stress concentrations display large secondary stress peaks, and so a need therefore exists for the ability to simultaneously reduce the amplitudes of multiple stress peaks occurring along a general boundary contour.

This report presents a new iterative shape optimisation method that has been developed for simultaneously minimising the peak tangential stresses on multiple segments around the boundary of a hole using finite element analysis. A comprehensive series of 2D and 3D numerical examples, some of which include significant geometric constraints, is explored in some detail and serves to demonstrate the efficacy of this new multi-peak shape optimisation method. It is shown that, when the optimal shape is achieved, in general the separate tensile and compressive stress segments occurring around the hole boundary converge to constant-stress regions of different values. The optimal hole shapes produce significant reductions in peak stress for all regions around the hole boundary, as compared to typical non-optimal circular holes. As the most fatigue critical location in a structure may not necessarily be the one with the biggest active tensile peak, it is desirable to be able to minimise these other stress peaks around the hole boundary. Hence, optimal shapes computed using the multi-peak method can usefully be applied to critical structural regions where it is usually highly desirable to substantially increase or maximise the fatigue life.

The multi-peak shape optimisation techniques developed and implemented by DSTO and reported here have improved the capability for rapid and effective creation of rework shapes for use in the solution of advanced structural shape optimisation problems in order to reduce the amplitudes of peak stresses. Using this capability, it will be possible to significantly enhance the fatigue lives of RAAF airframe structural components. Included in this report are sets of useful transferable coordinates of optimised shapes for selected geometry and loading cases typically occurring in aircraft structures. The computer program code that was developed and used to determine the optimal shapes is also provided, with some accompanying documentation, including an example input deck related to one of the hole geometries that was optimised.

Authors

Witold Waldman
Aerospace Division

Mr Witold Waldman completed a BEng (with distinction) in Aeronautical Engineering at the Royal Melbourne Institute of Technology in 1981. He commenced work in Structures Division in 1982 at what was then the Aeronautical Research Laboratory. He has published a number of papers and reports, and his experience has focussed on stress analysis using finite element and boundary element methods, structural mechanics, fracture mechanics, computational unsteady aerodynamics, structural dynamics testing, digital filtering of flight test data, nonlinear optimisation, and spectral analysis. His recent work has been in the areas of structural shape optimisation and computation of stress intensity factors. He is currently a Senior Research Engineer in the Structural and Damage Mechanics Group in the Airframe Technology and Safety Branch of Aerospace Division at DSTO.

Manfred Heller
Aerospace Division

Dr Manfred Heller completed a BEng (Hons) in Aeronautical Engineering at the University of New South Wales in 1981. He commenced employment in Structures Division at the Aeronautical Research Laboratory in 1982. He was awarded a Department of Defence Postgraduate Scholarship in 1986, completing a PhD at Melbourne University in 1989. He is currently the Group Head for Structural and Damage Mechanics in the Airframe Technology and Safety Branch of Aerospace Division at DSTO. Since 1992 he has led tasks that develop and evaluate techniques for extending the fatigue life of ADF aircraft components. His research contributions have focussed on the areas of structural shape optimisation, computational and experimental stress analysis, and bonded repair technology. He is a Corresponding Member on the Committee of Stress Analysis and Strength of Components, Engineering Science Data Units (ESDU) UK.

Contents

Nomenclature

b	distance between centres of two semicircular arcs
$\{\mathrm{B}_i\}_{i=1}^{q}$	q mesh generation blocks
c	an arbitrary characteristic length
d_i^j	increment of movement of the i-th node in the j-th stress subregion around the hole boundary
e	distance of centre of hole from edge of plate
E	Young's modulus
h	half-height of hole in plate
H	height of plate
K_t	stress concentration factor
N_1, N_2	parameters defining subdivision of automatic mesh generation blocks
r	radius of semicircular arc
s	step size scaling factor
S_1	applied remote stress in x-direction
S_2	applied remote stress in y-direction
u	arc length around perimeter of hole boundary
u_T	total arc length around the hole perimeter
w	half-width of hole in plate
W	width of plate
x, y	coordinates of orthogonal axis system
Γ	denotes the general hole boundary
μ	normalised arc length parameter = u/u_T
μ_T	total combined normalised arc length of constant stress zones around the entire hole boundary
θ	angle of inclination from the vertical
ρ	local radius of curvature
ρ_{min}	minimum radius of curvature
σ_i	local tangential stress at the i-th node
σ_i^j	tangential stress at node i in the j-th stress subregion around the hole boundary
σ_{th}^j	stress threshold corresponding to peak positive or negative stress occurring in the j-th stress subregion around hole boundary
$\{\sigma_{pj}\}_{j=1}^{n}$	n stress peaks around hole boundary
$\{\sigma_{pj}^*\}_{j=1}^{n}$	n zones of uniform stress
υ	Poisson's ratio

1. Introduction

Prior work has been undertaken at the Defence Science and Technology Organisation (DSTO) to determine precise optimal rework shapes. These aim to minimise the peak stresses occurring at existing holes and other stress concentrations in ageing aircraft structures, leading to substantial increases in fatigue life (Heller *et al.* 1999; Waldman *et al.* 2001; Waldman *et al.* 2002a; Heller *et al.* 2002; Burchill and Heller 2004a; Waldman *et al.* 2003, Burchill and Heller 2004b; McDonald and Heller 2004; Waldman and Heller 2005). The shapes were computed using a fully-automated iterative shape optimisation algorithm utilising finite element analysis (FEA). The method was implemented as a custom-written FORTRAN program executed via a shell script (Heller *et al.* 1999).

The shape optimisation algorithm used here is based on the biological growth analogy consistent with work of Heywood (1969) and Mattheck and Burkhardt (1990), where material is added where stresses are high and removed where they are low. For example, Heywood used a photoelastic technique to adjust locally (via material removal only) the boundary of an unsatisfactory design to obtain a uniform stress along the boundary of a stress concentrator, reducing significantly the stress concentration factor K_t values for optimised fillet profiles in the Rolls-Royce "Merlin" aero engine. In later work, Neuber (1972) provided a theoretical treatment showing that it is possible to alter locally the boundary shape of a stress concentrator to produce a uniform state of stress with a reduced value of K_t. This constant-stress feature of optimal shapes was also subsequently confirmed by Wheeler (1976), and early attempts at analytical approaches that had been formulated using this aim were those of Baud (1934) and Lansard (1955) in relation to fillets.

The shapes resulting from application of the DSTO shape optimisation method are fully free form and do not rely on analytical functions to represent the boundary shape, and the method was extended to include radius of curvature constraints (Waldman *et al.* 2002a). To date, the fully-automated single-peak approach has successfully reduced the dominant stress peak (tensile or compressive) on boundary segments of stress concentrators, producing more limited stress reductions in other regions (e.g. those having peaks of opposite sign). However, because the most fatigue critical location (i.e. the one that has the most rapid crack growth, and governs the choice of inspection interval) may not be the one with the biggest active tensile peak, it is also desirable to minimise these other stress peaks around the hole boundary. Key factors for different locations on the hole boundary segments are: (i) different residual stresses, both in peak values and stress distributions; (ii) the rate of stress decay normal to the hole boundary; and (iii) the probabilities of detection of cracks (i.e. a bigger crack size may be required for reliable detection of cracking at a secondary location). This situation occurred in service for fatigue-critical holes in the wing pivot fitting of the General Dynamics F-111 aircraft (Heller *et al.* 2002).

For a highly-idealised loading and unconstrained geometry, Vigdergauz and Cherkayev (1986) have shown analytically that an optimal shape with minimum K_t is obtained when the distribution of tangential stress around the boundary of a stress concentrator is piecewise constant. It is this fundamental concept that forms the basis for the DSTO multi-peak shape optimisation method. In the present work, we have fully automated the multi-peak method for the case of realistic highly geometrically constrained holes, to simultaneously minimise

the tangential stresses on multiple segments around a hole boundary. The advanced multi-peak method is described below, followed by a number of benchmark problems and other numerical examples, which together serve to demonstrate its efficacy and general applicability to a wide range of targeted problems. This comprehensive set of benchmarks is explored in some detail for a number of reasons.

Although a number of investigators have proposed "optimal" shapes (Heller 1969; Durelli and Rajaiah 1979; Dhir 1981;Vigdergauz and Cherkayev 1986), which are derived from semi-analytical, FEA or experimental methods, when these shapes are examined more closely (see Section 2.1), there are a number of issues with these solutions. These include: (i) when viewed in the light of the evaluation criteria proposed in this paper, their shapes are suboptimal in terms of their ability to maximise the reduction of stress concentrations; (ii) some solutions present K_t values but do not provide any shapes that can be verified; and (iii) many of the results are for highly-idealised problems and as a result are not accurately transferable to actual structures.

A description of the general multi-peak method and its numerical algorithm and implementation using a finite element analysis code is provided in Section 2. This is followed in Section 3 by the analysis of a series of benchmarks for comparison with idealised solutions available in the literature. These involve holes of various aspect ratios in plates that are remotely loaded uniaxially or biaxially. Further numerical examples involving holes near the edge of a plate, as well as ones for an inclined slotted hole, are presented in Section 4. Both types of holes include significant geometric constraints. An extension of the multi-peak method to 3D shape optimisation is covered in Section 5. Some general guidelines for performing multi-peak shape optimisation are provided in Section 6. An effective procedure for modifying sharp-cornered optimal hole shapes to improve robustness is described Section 7.

2. Description of method

2.1 Fundamentals of the multi-peak approach

To illustrate the DSTO multi-peak method, consider a hole in a remotely loaded large plate, as shown in Figure 1. Here u is the distance around the perimeter of the hole boundary Γ, and ρ is the local radius of curvature. If we define u_T to be the total arc length around the hole perimeter, then we can define a normalised arc length parameter $\mu = u/u_T$, such that $0 \leq \mu \leq 1$. There are k nodes distributed around the hole boundary, and the local tangential stress at the i-th node is σ_i. The optimisation is subject to an arbitrary geometric constraint consisting of a polygon with m sides and a minimum radius of curvature constraint $\rho \geq \rho_{min}$. The latter constraint can be activated to ensure that sharp corners, which are commonly found in some optimal shapes (Waldman *et al*. 2001; Waldman *et al*. 2002a; Vigdergauz and Cherkayev 1986; Durelli and Rajaiah 1979), do not develop. This results in optimal shapes that are more robust by being less sensitive to variations in loading direction, as well as being more amenable to manufacture. However, this can result in a relatively small but clear trade-off with regard to the degree of reduction in peak stresses that can be achieved under these circumstances (as shown in Figure 2 and Figure 10 and discussed later).

The typical multi-peak distribution of tangential stress around the boundary of a circular hole in a uniaxially-loaded plate is shown in Figure 2. The zero crossings in the stress distribution are used to identify the set of n subregions of positive and negative stress. In Figure 2 there are n = 4 distinct stress peaks around the closed boundary, corresponding to the n = 4 zero crossings. Each of the stress peaks is denoted by $\{\sigma_{pj}\}_{j=1}^{n}$. In general, for non-circular holes, the peaks would not be expected to be equal to one another. Consistent with the work of Vigdergauz and Cherkayev (1986), as well as prior non-automated multi-peak shape optimisation work by Durelli and Rajaiah (1979), we postulate that a constrained optimal shape will have n stress peaks consisting of zones of uniform stress, denoted in Figure 2 by $\{\sigma_{pj}^{*}\}_{j=1}^{n}$. We note in passing that the optimal solutions to many problems will have multiple stress peaks, while some will reduce to an optimal shape with one stress peak only (e.g. the classic case of a biaxially-loaded plate with an optimal elliptical hole, where the aspect ratio of the optimal ellipse is equal to the ratio of the remote stresses, which are of the same sign; or the case of two closely-spaced interacting holes (Cherepanov 1974; Waldman, Heller and Rose 2003; Vigdergauz 2008; Vigdergauz 2010)).

In our FEA implementation, using a custom-written FORTRAN 90 program (the source code listing is provided in Appendix A), during each iteration the stresses are computed at all the nodes around the hole boundary. The i-th boundary node, which is located in the j-th stress subregion, is then moved in the direction of the local outward normal to the boundary by a distance d_i^j. This distance is computed using

$$d_i^j = \left(\frac{\sigma_i^j - \sigma_{th}^j}{\sigma_{th}^j} \right) sc, \quad \sigma_{th}^j = \max(\sigma_i^j) \text{ if } \sigma_i^j > 0 \text{ or } \sigma_{th}^j = \min(\sigma_i^j) \text{ if } \sigma_i^j < 0, \; j = 1, 2, \ldots, n \qquad (1)$$

where σ_i^j is the tangential stress at node i in the j-th stress subregion, σ_{th}^j is the stress threshold corresponding to the peak positive or negative stress occurring in the j-th stress subregion, c is an arbitrary characteristic length, and s is a step size scaling factor. Due to the selection of the stress threshold, in all cases we are only removing material, so the optimal shape will encompass the initial starting shape, which is advantageous in the context of shape reworking of an existing airframe component.

In general, for realistic multi-peak problems subject to geometric constraints, the final optimal shape will have some dependence on the size and the aspect ratio of the initial shape and the prevailing geometric constraints. This issue is discussed in more detail in the numerical examples.

2.2 Convergence checks and evaluation of solution quality

The process described above is repeated until the stresses in each subregion converge to a reduced constant value, to within some prescribed tolerance, over the maximum possible length of arc (maximising the length of the constant stress region is well known as leading to local stress minimisation (Neuber 1972)). For typical complex constrained problems, each segment converges to a flat-topped region of constant stress of different magnitude to give the best overall solution. There are usually no distinct flat-topped stress regions present initially, but these soon develop as the solution converges towards the optimal shape. The length of each such zone of uniform stress, where the stress is within some nominal

percentage of the peak value in the j-th stress subregion (1% was used for the work presented here), is denoted by $\{\mu_j\}_{j=1}^{n}$ (see Figure 2 for a typical case with n = 4). The quality of the solution is assessed at each step by computing the length of each of these zones, from which

$$\mu_T = \sum_{j=1}^{n} \mu_j \qquad (0 \le \mu_T \le 1) \tag{2}$$

can be determined. Here μ_T represents the total combined length of the constant stress zones around the entire boundary of the hole. The parameter μ_T is equivalent to the quantitative coefficient of efficiency first proposed by Durelli, Brown and Yee (1978). As the solution shape approaches optimality and the stress peaks are reduced, the individual lengths of these flat-topped stress zones progressively increase. When the optimal solution has been obtained, it is found that μ_T has reached a maximum for the problem under consideration and the magnitude of the peak stress in each stress subregion has been minimised. In the limiting case, where there are no geometric constraints invoked, it is expected that the condition of μ_T = 1 will be obtained. The range and standard deviation of the stresses in the flat-topped stress zones are also computed, and they are used in the assessment of solution optimality, wherein small values of these parameters are indicative of a high-quality converged optimal solution.

2.3 FEA code used and element types

All of the FEA work described in this paper was performed on a Hewlett-Packard K260 Series 9000 computer and utilised the commercially available PAFEC FE Level 8.5 FEA code. This code has also been used in prior shape optimisation work. The PUPPIES utility program was used for extracting stress data from the PAFEC binary output files. All the 2D FEA meshes consisted of 8-noded quadrilateral isoparametric elements and linear elastic plane stress conditions were assumed. The 3D FEA meshes utilised 20-noded isoparametric solid elements. The material properties used were typical of those for an aluminium alloy, with Young's modulus E = 73 GPa and Poisson's ratio υ = 0.32.

2.4 Automated mesh updating normal to stress concentrator boundary

When the shape changes as a result of material removal, the finite element mesh must be adjusted in order to avoid mesh distortion. The approach that has been used in conjunction with the PAFEC FE code is explained below.

By using a valuable and convenient automatic mesh generation feature that is available within PAFEC, which is called PAFBLOCKS, only the nodal movements at the stress concentrator boundary need to be specified. For two-dimensional FEA, the mesh generation blocks are usually four-sided subregions of the FEA model consisting of a number of plate elements. Figure 3a shows the general idealised layout of an FEA mesh that makes use of q mesh generation blocks, $\{B_i\}_{i=1}^{q}$. Figure 3b shows a detail view of two such mesh generation blocks, which are labelled as B_1 and B_2. Using two parameters, N_1 and N_2, each mesh generation block can be subdivided into a number of smaller elements in the spanwise and chordwise directions as shown in the diagram. The parameters N_1 and N_2 define how many

elements are to be created within each mesh generation block, as well as the spacing of those elements. In order for proportionately sized internal elements to be created, only the locations of the corner nodes and the number of elements along the sides of the blocks need to be specified. The thick dotted line in Figure 3b denotes the moving boundary during the shape optimisation, while the edges of the elements within each mesh generation block are denoted by the thin dotted lines. For each mesh generation block, the edge of the block that lies directly opposite to the moving boundary remains fixed, with the thick solid line denoting the fixed boundary.

Within the PAFEC code, the edges of each mesh generation block are defined by two corner nodes and a midside node in such a way that these three nodes lie on a circular arc or a straight line segment. The former applies for the initial FEA mesh for a model of a circular hole. These multiple circular arcs are used by PAFEC for the generation of intermediate nodes. At every iteration during the shape optimisation, the nodes on the free boundary that define the edges of each of the mesh generation blocks are completely free to move to any location in accordance with Equation (1). The internal elements within each of the mesh generation blocks simply shrink or grow in proportion to this nodal movement. Consider the case where there are n element corner nodes arbitrarily spaced along a typical side of a mesh generation block that consists of a straight edge. At the m-th iteration, the coordinates (x_j, y_j) of the j-th node are as shown in Figure 3. If the predicted movement of the boundary node (x_1, y_1) is denoted by $d_1 = (\Delta x_1, \Delta y_1)$, then the new coordinates of the q element corner nodes at iteration m+1 can be computed using

$$x_j^{m+1} = x_j^m + \Delta x_1^m \left(\frac{x_j^m - x_q^m}{x_1^m - x_q^m} \right) \tag{3}$$

$$y_j^{m+1} = y_j^m + \Delta y_1^m \left(\frac{y_j^m - y_q^m}{y_1^m - y_q^m} \right) \tag{4}$$

The automated remeshing of elements by the PAFBLOCKS helps to maintain mesh integrity by avoiding mesh distortion, something that can cause significant problems for other FEA-based shape optimisation programs. To some extent, the selection of the number of segments to be used to subdivide and model the moving boundary is arbitrary. However, if there are not enough segments then this will be readily apparent in the solution stresses, which will not be uniform along the boundary of the hole as required for a true optimal solution.

2.5 Complex geometric constraints

The optimisation procedure implemented here has provision for enforcing geometric constraints on the boundary of the hole as it evolves during the shape optimisation process. The two types of constraints that the hole is not permitted to violate can include a minimum radius of curvature constraint and an arbitrary multi-sided polygonal constraint boundary. These constraints can be applied individually or together, as determined by the user, to suit the requirements of the problem under consideration.

During the shape optimisation process, a new set of node locations is computed at every iteration, thus defining a new boundary for immediate potential use. The radius of curvature is calculated at each nodal point around this new hole boundary using a three-point circle curve fit, which also conveniently handles straight-line segments that may often be present. Each of the new node locations is also checked to see if it lies within the polygonal boundary constraint. This is done using an algorithm proposed by Morris (1993), which can determine whether a point lies within a polygon. If either of these two constraints is violated, then the last calculated normal displacement of each offending node is repetitively halved, until neither constraint is violated anywhere around the boundary. Although simple in its approach, this algorithm works well in practice, maintaining a smooth hole boundary. It should be noted that many problems do not require the use of a multi-sided polygonal bounding box constraint.

2.6 Automated mesh updating tangential to stress concentrator boundary

As the shape changes because of material removal, the tangential spacing of the elements around the stress concentrator boundary must often be adjusted to avoid mesh distortion. The approach that has been used is explained below.

At each iteration, once a valid hole boundary that does not violate the geometric constraints has been determined, a parametric interpolant consisting of a spline under tension is fitted to the new node coordinates on the boundary. For small tension factors the spline under tension resembles a cubic spline, while for large tension factors it resembles a piecewise linear interpolant. The theoretical development of a spline under tension is given by Cline (1974a), and the source code to the FORTRAN subroutines is provided by Cline (1974b). Note that electronic copies of the subroutines were obtained from the FITPACK library of FORTRAN subroutines available from the publicly accessible NETLIB software repository, which is described by Dongarra and Grosse (1987).

Its smoothness and ability to accurately reflect local behaviours make the spline under tension an excellent candidate for accurately representing the 2D free-form boundary curve at each step of the optimisation. The spline is fitted through a sequence of k points (commonly known as knots), $\{(x_i, y_i)\}_{i=1}^{k}$, corresponding to each of the nodes around the hole boundary, and a mapping of points in the interval [0, 1] onto a curve in the plane can be performed. The use of both open and closed parametric curves is fully supported, and a zero tension factor was utilised for the interpolation. The ability to accurately interpolate a closed curve is important for conducting shape optimisation of holes in complex structural geometries and loadings where conditions of symmetry cannot be used to advantage, necessitating the use of a full FEA model.

As implemented in the multi-peak shape optimisation method, the parametric spline curve-fit allows the nodes along the boundary to be automatically redistributed, and it is well suited to handling the large number of nodal boundary points that typically feature in applications of gradientless shape optimisation methods. The parametric curve-fit enables any predefined mesh spacing to be retained throughout the optimisation, and it is often convenient to choose to maintain the mesh spacing that was used for the initial shape. The application of this node redistribution strategy at every iteration circumvents problems with

mesh distortion, which would otherwise require cumbersome and time-consuming manual intervention to carry out remeshing one or more times before the optimal shape is obtained. Significant mesh distortion is an area of considerable concern in other shape optimisation codes. It becomes a major issue when it is necessary to compute optimal shapes that contain sharp corners, to the extent that most codes simply fail to converge to a sharp-corner optimal shape. The use of parametric spline curve-fits and the redistribution of the nodes around the boundary work well in tandem to prevent mesh distortion from arising in such instances.

By coupling the node redistribution algorithm with the element generation capability provided by the PAFBLOCKS feature of the PAFEC FE code, an automated remeshing capability that is both elegant and adaptable is embodied in the multi-peak shape optimisation code. This allows smooth as well as sharp-cornered optimal shapes to be easily and reliably obtained for a wide range of typical problems, particularly those that involve very large shape changes between the initial and the final optimal shapes. There is minimal computational overhead, and the method copes easily with the enforcement of minimum radius of curvature and arbitrary polygonal geometric constraints.

It is important to note that the use of the splines does not change the optimal shape. The shape depends simply on the predicted movements, the minimum radius of curvature constraints, and any multi-sided polygonal bounding box constraints (if activated).

3. Comparison with idealised solutions

To evaluate the performance of the multi-peak shape optimisation method in determining the shapes of optimal holes, we considered some classic cases of holes of various aspect ratios in plates under uniaxial and reversed biaxial loading. The results of the multi-peak method for these highly-idealised problems were compared to the limited solutions that were available in the literature. Although they have limited practical applicability for complex structures and loadings, it is nevertheless instructive to consider them as benchmark cases. The general notation and geometry for a central hole in a biaxially-loaded plate is shown in Figure 4, where the plate width is W and the plate height is H. Here the initial central hole is depicted as an ellipse whose semi-major axis is h and semi-minor axis is w, representing a hole of aspect ratio h:w. The dimensions of the plate were chosen to be relatively large in comparison to the hole size in order to reduce the effects of finite plate size on the results (W/w = 10, and H/h = 5 when h:w = 2 or H/h = 10 when h:w = 1), therefore allowing better comparison with infinite plate solutions where available. S_1 and S_2 are the uniform in-plane remote stresses applied to the plate in the x- and y-directions, respectively. Figure 5 shows the local detail of the ¼-symmetry FEA mesh for a typical optimal shape.

3.1 Central hole in 2D plate under uniaxial loading

Multi-peak and single-peak shape optimisations were performed for the case of uniaxial tensile loading in the y-direction (i.e. $S_2 > 0$, $S_1 = 0$) and hole aspect ratios of h:w = 1:1 and 2:1. In the FEA, a uniform uniaxial in-plane tension load S_2 = 100 MPa was applied to the plate. Values of $c = h$ = 100 mm and s = 0.005 were used for the optimisation in accordance with Equation (1). Table 1 includes the results of analyses of two distinct aspect ratio cases that were studied to provide data for typical minimum radii of curvature that are of practical

interest. The minimum radius of curvature ρ_{min} was constrained to ρ_{min}/u_T = 0.02 and 0.04, where u_T is the total circumference around the hole. There are n = 4 stress peaks around the hole boundary, denoted by stress concentration factors $\{K_{tj}\}_{j=1}^{4}$, and due to symmetry we have that $K_{t1} = K_{t3}$ and $K_{t2} = K_{t4}$. Also given in Table 1 is the normalised length $\{\mu_j\}_{j=1}^{4}$ of each distinct flat-topped stress zone where the stress lies within 1% of the peak value, together with the total normalised length μ_T of the hole boundary meeting this criterion.

3.1.1 Discussion of 1:1 aspect ratio case

For ρ_{min}/u_T = 0.02, the value of the peak tensile K_t obtained for this case using the multi-peak method (K_{t1} = 2.193) is negligibly higher (0.14%) than that obtained using the single-peak method (K_{t1} = 2.190). However, the multi-peak method was able to significantly reduce the size of the compressive stress peak relative to the reduction produced by the single-peak method. We note also that K_{t1} and K_{t2} decrease as ρ_{min}/u_T decreases, and that ρ_{min}/u_T = 0.02 represents only a relatively moderate minimum radius of curvature. Correspondingly, as ρ_{min}/u_T decreases, the lengths of the flat-topped stress zones increase quite considerably, with the largest gains being consistently achieved by the multi-peak method. This indicates that the optimality of the solution achieved by the multi-peak method is better than that produced by the more limited single-peak approach.

It is a general feature of optimal shapes that the reduction in the values of the peak stresses that can be achieved becomes less as the allowable minimum radius of curvature increases. This also brings about a reduction in the total length of the constant-stress regions, indicating that the solution will not be as optimal as one with a tighter minimum radius of curvature.

Table 2 presents the transferable ¼-symmetric normalised coordinates (x/w, y/w) for optimal hole shapes for two representative cases with aspect ratio h:w ≈ 1:1. The shapes for minimum radius of curvature constraints of ρ_{min}/w = 0.150 and ρ_{min}/w = 0.286 are provided.

A number of other researchers have also attempted to determine the optimal shape of a hole of 1:1 aspect ratio in a uniaxially-loaded plate, and their results are presented in Table 1. Durelli and Rajaiah (1979, 1980, 1981) used a two-dimensional photoelastic shape optimisation technique to determine a solution to this problem. They placed particular emphasis on obtaining a uniform stress distribution around the discontinuity even in the presence of reversal of stress around the hole, an approach that is conceptually similar to the present multi-peak shape optimisation method that is implemented here. By conducting the optimisation for both the tensile and the compressive segments around the hole boundary, the optimal shape that they obtained was that of a quasi-square hole (i.e. a square hole with rounded corners). Their optimal shape produced a peak tensile K_t = 2.54 and a peak compressive K_t = –0.87 (for W/w = 7.1), and the tangential stresses tended to remain uniform along large portions of both the tensile and compressive segments around the edge of the hole.

The optimal shapes for some solutions are compared in Figure 6a, which also gives the starting location for the measurement of the arc length u around the hole boundary. For the single-peak optimal shape, note that the vertical extent of the shape was constrained during the optimisation process so as not to exceed that which had been defined by the multi-peak

optimal shape that had been computed first. This was done to ensure that the aspect ratios of the single-peak and multi-peak shapes were identical to allow comparison of their resultant K_t values. The corresponding K_t distributions are shown in Figure 6b, together with the results obtained by Durelli and Rajaiah (1981) that have a peak tensile K_t = 2.54. However, as indicated by an FEA of the optimal shape of Durelli and Rajaiah (1981) that was carried out by Schnack (1983), the photoelastic technique appears to have difficulty resolving small stress variations. Schnack determined an optimum minimised peak tensile K_t = 2.33, which is still higher than that achieved by the present multi-peak method (K_t = 2.19), even though the minimum radius of curvature associated with Schnack's solution was much smaller. Furthermore, the length of the boundary region over which the stresses are uniform was also significantly less than that obtained using the present multi-peak method (μ_T = 0.55 versus μ_T = 0.90).

Looking further at the optimal shapes presented in Figure 6a, yet another feature is that, in the central portion of the constant-stress region corresponding to an optimised stress peak, the edges of the boundaries have a large radius of curvature, often tending towards being a straight line. It is also interesting to note that the sides are convex or concave depending on the sign of the tangential stress, the latter being obtained when the multi-peak method is used. However, it should be noted that the allowable minimum radius of curvature must be sufficiently small to enable convex sides to form.

Looking further at the K_t distributions shown in Figure 6b, it is evident that along the hole boundary there is a very rapid step-like change from a region of uniform positive tangential stress to one of negative tangential stress, and vice versa. This rate of change of stress is much greater than that which occurs for the initial unoptimised shape (in this case a circular hole). It should be noted that the width of the transition region going from positive to negative stress increases with increasing minimum radius of curvature, ρ_{min}. As the value of the minimum radius of curvature constraint is increased, the corners that are present in the plot of K_t versus μ will become more rounded, and the maximum value of the rate of change of K_t in the transition region will decrease.

In other work, Rajaiah and Naik (1983) presented an analytical expression for computing the tangential stress around the boundary of a quasi-square or quasi-rectangular hole in an infinite biaxially-loaded plate. Their solution made use of conformal transformation and an analytical expression to define the parametric shape of the quasi-square hole that had rounded corners and sides. This allowed the peak K_t value to be computed for a range of geometries in order to determine the minimum peak K_t and the corresponding hole shape. For their optimum quasi-square hole solution, corresponding to ρ_{min}/u_T = 0.074, they obtained a peak tensile K_t = 2.469 and a peak compressive K_t = –0.911. However, with a very low value of μ_T = 0.158, they were unable to reproduce the same degree of stress uniformity obtained by Durelli and Rajaiah (1980), whose optimal shape had μ_T = 0.655. This indicates that the analytical solution proposed by Rajaiah and Naik (1983) is somewhat suboptimal and can therefore be improved upon. In comparison, the optimal shape computed by Schnack (1983) had μ_T = 0.552, which is comparatively much closer to that obtained by Durelli and Rajaiah (1980).

The results obtained by Rajaiah and Naik (1983) have been checked during the current study using a finite element model to compute the tangential stress around the boundary of the quasi-square hole in a large plate of finite width and height ($H/h = W/w = 10$). The overall distribution of tangential stress was very similar to that obtained by Rajaiah and Naik (1983), with the present FEA estimating a peak tensile K_t = 2.54 and peak compressive K_t = –0.97, which are 2.8% and 6.6% higher than their published results. In comparison, a circular hole in our particular finite plate has a peak tensile K_t = 3.09 and peak compressive K_t = –1.07, which are 3.0% and 7.0% higher than the exact analytical results for a circular hole in an infinite plate. Once these intrinsic differences between the infinite-plate and finite-plate solutions have been taken into account, it can be concluded that the solution offered by Rajaiah and Naik (1983) is actually in very good agreement with that obtained by FEA.

Dhir (1981) also analysed this case by minimising an integral involving the stresses around the hole boundary, obtaining an optimal shape with a peak tensile K_t = 2.47 and a peak compressive K_t = –0.92. This solution corresponded to a value of $\rho_{min}/u_T \approx 0.15$ (determined by scaling from the given plot of the optimal shape), which is a relatively large minimum radius of curvature. The optimal shape was of 1:1 aspect ratio and had rounded corners and sides, and it was very similar to the shape obtained by Rajaiah and Naik (1983). Dhir (1981) noted that his proposed shape optimisation method leads to a relatively constant boundary stress distribution with attenuated peaks. From the stress plot that was provided, this indeed appeared to be the case, but the lengths of what might be assessed as being constant-stress zones were also quite short, indicating that the proposed solution was still in fact suboptimal. Note that the peak tensile K_t value in the solution obtained by Dhir (1981) is effectively identical to that obtained by Rajaiah and Naik (1983) for ρ_{min}/u_T = 0.074, and the peak compressive K_t is also in very close agreement.

Inspection of the results for uniaxial loading presented in Table 1 indicates that the multi-peak method produces optimal shapes that have superior stress concentration reducing properties. For the hole of nominal 1:1 aspect ratio and ρ_{min}/u_T = 0.02, a peak tensile K_t = 2.19 and peak compressive K_t = –0.83 were obtained. The fact that μ_T = 0.900 indicates that a highly optimal solution has been achieved by the present multi-peak method, and it is noted that the tensile constant-stress zone was moderately shorter than the compressive constant-stress zone. Although the single-peak shape optimisation for this case produced effectively the same peak tensile K_t, the peak compressive K_t = –0.89 was about 7.5% higher.

3.1.2 Discussion of 2:1 aspect ratio case

For the case of a hole of nominal 2:1 aspect ratio, which has a lower stress concentration effect than a 1:1 hole, the multi-peak method produced optimal shapes that had lower peak stresses than the shape proposed by Rajaiah and Naik (1983). The multi-peak results were also better than those offered by the single-peak method, both in terms of reducing the magnitude of the peak compressive stress and the total extent of the zones of uniform stress.

All the analyses that were undertaken indicate that the "optimal" hole shapes depend on the aspect ratio of the initial starting shape. The numerically determined optimal shapes effectively have the same aspect ratio as the starting shape, but they have increased area and reduced stress. Overall, inspection of the results in Table 1 shows that, for any given hole

aspect ratio and loading case, as $\rho_{min}/u_T \to 0$ then $\mu_T \to 1$. The only points on the boundary that are not in a constant stress zone are those that have locally invoked the ρ_{min} constraint. As expected, increasing the aspect ratio of the hole provides progressively lower values of the peak stresses.

Table 2 also presents the transferable ¼-symmetric normalised coordinates (x/w, y/w) for optimal hole shapes for two representative cases with aspect ratio h:$w \approx$ 2:1. The shapes corresponding to minimum radius of curvature constraints of ρ_{min}/w = 0.218 and 0.422 are provided.

3.2 Central hole in 2D plate under reversed biaxial loading

Two different cases involving a hole in a plate under reversed biaxial remote loading were also investigated. The first considers holes with a 1:1 aspect ratio under loading conditions where the biaxial loading is of equal and opposite sign, S_2:S_1 = 1:–1. The second looks at holes with 2:1 and 1.329:1 aspect ratios under loading conditions where the biaxial loading is of S_2:S_1 = 1.377:–1.

Table 3 shows the results obtained using the multi-peak method for a hole of 1:1 aspect ratio (in a plate of W/w = 10) where the large plate is loaded remotely by the reversed biaxial loading S_2:S_1 = 1:–1. A variety of normalised minimum radius of curvature constraints were considered in the range $0 \le \rho_{min}/w \le 0.500$. Table 4 shows the results for the optimised benchmark problems that involved a reversed biaxial loading of S_2:S_1 = 1.377:–1, which were conducted for holes with aspect ratios h/w = 1, 1.329, and 2. For optimised holes with aspect ratio h/w = 2, minimum radius of curvature constraints of ρ_{min}/w = 0.222 and 0.430 were used. For the optimised holes with aspect ratio h/w = 1.329, minimum radius of curvature constraints of ρ_{min}/w = 0.100 and 0.174 were used. For the optimised hole with aspect ratio h/w = 1, a minimum radius of curvature constraint of ρ_{min}/w = 0.100 was used.

3.2.1 Discussion of S_2:S_1 = 1:–1 loading case

Figure 7a presents the optimal shapes for the multi-peak solutions for ρ_{min}/u_T = 0 and 0.02, and compares them against the double barrel hole (with W/w = 6.13) that Durelli and Rajaiah (1981) thought would be close to the optimal shape, as well as the initial circular hole. It also gives the starting location for measurement of the arc length u around the hole boundary. The corresponding K_t distributions are shown in Figure 7b, where the rounded peaks of the double barrel hole are clearly contrasted against the flat-topped zones produced by the multi-peak method. The peak K_t = 2.84 for ρ_{min}/u_T = 0 is considerably lower than the peak K_t = 3.58 for the double barrel hole. These two features clearly indicate that the double barrel hole is not an optimal solution for this load case. The shape of the suboptimal double barrel hole lies between that of the circular hole and the multi-peak optimal shapes. Its higher stress concentration is caused by the use of a relatively large value of minimum radius of curvature, with ρ_{min} = 0.23w in the corners, corresponding to ρ_{min}/u_T = 0.068.

It is noted that ρ_{min}/u_T = 0 corresponds to a solution with sharp corners, and this particular case is important as it represents the best possible optimal solution that can be achieved for this geometry and loading condition. It therefore serves as a baseline against which solutions

for other ρ_{min}/u_T values can be compared, even though it may be only of academic interest because of its sensitivity to changes in load orientation. In agreement with an observation reported by Durelli and Rajaiah (1979, 1980), we note here that, if sharp corners are allowed to occur in optimised shapes, they produce zero stress at the corner locations (i.e. the distribution of tangential stress around the hole boundary does not have any singularities, contrary to many an engineer's intuition). The $\rho_{min}/u_T = 0$ case is also a stringent test case for the DSTO shape optimisation program, as a sharp reentrant corner must be created and managed in order to obtain a solution. The sharp-cornered shape that has developed during the optimisation of the hole for the $\rho_{min}/u_T = 0$ case indicates that the program is able to handle cases such as this.

Looking further at the K_t distributions shown in Figure 7b, the very rapid transitions between the positive and negative regions of tangential stress are evident for the optimal shapes, just as they were in Figure 6b. In the limit, a step change is produced when the corner radius is permitted to go to zero, which corresponds to the case where $\rho_{min}/u_T = 0$. A zoomed-in view of one of the corner regions in the K_t distribution is provided in order to see this characteristic behaviour more clearly.

Using an approximate analytical solution for $\rho_{min}/u_T = 0$, Vigdergauz and Cherkayev (1986) have tabulated infinite-plate optimal K_t values for a small number of ratios of reversed biaxial loading $-S_2/S_1$ between 0 and 1. In their solution, the absolute value of K_t is constant around the entire hole boundary. Their results are plotted in Figure 8. There the optimal shape for the case $-S_2/S_1 = 0.866$ corresponds to a hole of aspect ratio $h/w = 1.47$, and the aspect ratio increases as $-S_1/S_2 \to 0$. Note that there is a step-like change in K_t near $-S_2/S_1 = 0.3$, which is not understood. Nevertheless, keeping this discrepancy in mind, our results were still compared to those obtained by Vigdergauz and Cherkayev (1986).

For the case where $S_2 = -S_1$, Vigdergauz and Cherkayev (1986) obtained a solution with $|K_t| = 2.732$. Although they did not provide a plot of the optimal shape, which has a 1:1 aspect ratio, they did mention that it differs from a square only in that small segments of the rectilinear boundary adjacent to each corner have become rounded. Application of the present free-form multi-peak optimisation method for $\rho_{min}/u_T = 0$ produced an optimal finite-plate solution with $|K_t| = 2.844$, which is only 4.1% higher than their infinite-plate solution. Importantly, the stresses produced by the multi-peak method were uniform around the entire hole boundary ($\mu_T = 1$), indicating that a fully optimal shape had indeed been achieved.

For the case where $S_2 = -S_1$, with $\rho_{min}/u_T = 0.04$, the multi-peak method has an optimal tensile $K_t = 3.021$ and compressive $K_t = -3.019$. This result is in quite close agreement with Rajaiah and Naik (1983), who obtained $|K_t| = 3.074$ for a slightly larger minimum radius of curvature $\rho_{min}/u_T = 0.047$. For this same loading case, Dhir (1981) obtained $|K_t| = 3.07$ for an infinite-plate solution that produced a square-like (double barrel shape) with rounded corners (with unknown ρ_{min}/u_T). For $\rho_{min}/u_T = 0.02$, the present multi-peak method produced an even greater reduction in peak stress, with an optimal tensile $K_t = 2.942$ and compressive $K_t = -2.940$. This result is only about 3.4% higher than for the sharp-corner optimal shape with $\rho_{min}/u_T = 0$.

Table 5 presents the transferable ¼-symmetric normalised coordinates (x/w, y/w) for optimal hole shapes for three representative cases with aspect ratio h:w = 1:1. The shapes are provided for cases with minimum radius of curvature constraints of ρ_{min}/w = 0, 0.150 and 0.286.

3.2.2 *Discussion of S_2:S_1 = 1.377:–1 loading case*

Looking at the results presented in Table 4 for the reversed biaxial loading of S_2:S_1 = 1.377:–1, it is noted that the peak stress is compressive for hole aspect ratios h/w = 1.329 and 2. For the hole with h/w = 2, the multi-peak optimal shapes produced flat-topped peaks of K_t = –3.020 and K_t = +2.424 for ρ_{min}/w = 0.111, and K_t = –3.199 and K_t = 2.465 for ρ_{min}/w = 0.215. For the hole with aspect ratio h/w = 1.329, the optimal shape produced flat-topped peaks with K_t = +2.561 and –2.560 for ρ_{min}/w = 0.087. It is interesting to note that the peak positive and negative K_t values were essentially the same in magnitude for this particular case.

In order to provide a point of comparison with shapes obtained using an alternative approach, the method proposed by Rajaiah and Naik (1983) was implemented in a FORTRAN program. Some of the numerical results that were produced are provided in Table 4, where it is quite clear that the K_t values that were obtained using Rajaiah and Naik's approach were generally greater than those produced by the multi-peak shape optimisation method. This is as a result of the constraints on the possible shapes that were imposed by the simple trigonometric functions used to define the behaviour of their assumed shape.

Table 6 presents the transferable ¼-symmetric normalised coordinates (x/w, y/w) for optimal hole shapes for the two different hole aspect ratios that were analysed. For the optimal holes with aspect ratio h:w = 2:1, the shapes for minimum radius of curvature constraints of ρ_{min}/w = 0.222 and 0.430 are provided. For the optimal holes with aspect ratio h:w = 1.329:1, the shapes for ρ_{min}/w = 0.100 and = 0. 174 are provided.

4. Numerical examples for other 2D cases

In the prior section, the performance of the multi-peak shape optimisation method was demonstrated by solving a number of idealised problems, some of which included a modest geometric minimum radius of curvature constraint. We will now proceed to showcase the capabilities of the multi-peak method when confronted with holes subject to progressively more demanding geometric constraints. The examples analysed here are more typical of the realistic practical problems that are often encountered in aircraft structures.

4.1 Large 2D plate with central hole near one edge

Figure 9a shows a hole located close to an edge in a large rectangular plate. A moderate geometric constraint is active to prevent the edge of the hole from getting closer to the plate edge during the optimisation. In this analysis, the height and width of the plate were H = 1000 mm and W = 600 mm, and the vertical constraint line was located at x = –50 mm. Three initial elliptical hole geometries were studied, both with the centre of the ellipse located a distance e = 100 mm from the edge of the plate. The first was a 1:1 ellipse (a circular hole) with h = 100 mm and w = 100 mm. The second was a 2:1 ellipse with h = 200 mm and w = 100

mm. The third was a 3:1 ellipse with h = 300 mm and w = 100 mm. The plate was subjected to a uniaxial in-plane tension stress of S_2 = 100 MPa applied to the plate edges at y = ±500 mm. A number of different cases of minimum radius of curvature were investigated for each configuration, with typical values being ρ_{min}/u_T = 0.02 and 0.04. Values of characteristic length c = 100 mm and step size s = 0.004 were used for the optimisations in accordance with Equation (1), and smooth convergence to an optimal shape was typically achieved after 70–150 iterations. Figure 9b shows the local detail of the ½-symmetry FEA mesh for a typical optimal shape corresponding to a hole with ρ_{min}/u_T = 0.02 and final aspect ratio 2.068:1.

Figure 10a shows a comparison of the shape of the initial 2:1 elliptical hole with that of the optimal hole for cases with ρ_{min}/u_T = 0.02 and 0.04, and it also gives the starting location for the measurement of the arc length u around the hole boundary. Figure 10b shows the corresponding variation of K_t around the hole boundary for the initial elliptical hole and the final free-form optimal holes. Between the four zero crossings, the initial 2:1 elliptical hole has n = 4 major stress peaks, which from left to right are $\{K_{tj}\}_{j=1}^{4}$ = {3.089, –1.509, 2.325, –1.509}. The optimal shape reduces these peaks to flat-topped zones with values of $\{K_{tj}\}_{j=1}^{4}$ = {2.270, –1.011, 2.030, –1.011}, which are all uniform to within a tolerance of 0.0005. This represents reductions in each of the respective peak stresses of 26.5%, 33.0%, 12.7% and 26.5%. The high degree of stress uniformity indicates excellent convergence in the optimal solution, and 90% of the circumference has regions with uniform stress (μ_T = 0.900). The aspect ratio of the optimal shape has increased only slightly from 2:1 to 2.068:1, although the corresponding increase in the area of the hole is more significant. This type of behaviour is a typical characteristic of the optimal shapes determined by multi-peak method. Note that for this particular case the vertical constraint line at x = –50 mm was in force only at the single point (x, y) = (–50, 0), although the shape approached this line quite closely for a large proportion of the left hand edge of the hole. The optimal shape with a larger minimum radius of curvature ρ_{min}/u_T = 0.04 produced flat-topped zones with values of $\{K_{tj}\}_{j=1}^{4}$ = {2.319, –1.054, 2.066, –1.054}, which are slightly lower reductions in the peak stresses of 24.9%, 30.2%, 11.1% and 30.2% than before.

The overall set of results for optimal holes close to an edge of a plate is presented in Table 7, which gives the values of $\{K_{tj}\}_{j=1}^{4}$ and proportions $\{\mu_j\}_{j=1}^{4}$ for each of the four optimised regions with constant stress. For ease of comparison, K_t values for the initial starting shapes are also provided in the bottom part of this table. It is also seen that the aspect ratios of the optimal shapes hardly change from that of the initial shapes, even though there are substantial stress reductions; what changes is the area of the hole. As expected, for a given hole aspect ratio and loading case, we find that as $\rho_{min}/u_T \to 0$ then $\mu_T \to 1$. It is noted once again that the only points on the boundary that are not in a constant stress zone are those that have locally invoked the ρ_{min} constraint (i.e. in the corner regions of the optimal shape shown in Figure 10a).

Figure 11 shows the optimal shapes that were obtained using multi-peak shape optimisation for a hole of aspect ratio h:w ≈ 1:1 close to an edge (e/w = 2) in a uniaxially-loaded large rectangular plate with minimum radius of curvature constraints ρ_{min}/w = 0.200 and 0.400. Figure 12 shows the optimal shapes obtained from multi-peak shape optimisation for a hole of aspect ratio h:w ≈ 3:1 close to an edge (e/w = 2) in a uniaxially-loaded large rectangular

plate for ρ_{min}/w = 0.300 and 0.570. As expected for this uniaxial loading case, the amplitudes of the peak K_t values decrease with increasing hole aspect ratio h:w.

Three sets of transferable ½-symmetric normalised coordinates (x/w, y/w) for optimal hole shapes for the case of a hole close to an edge (e/w = 2) in a uniaxially-loaded large plate have been computed and are provided here. Table 8 presents the optimal shapes for a hole aspect ratio of h:w ≈ 1:1 and minimum radius of curvature constraints of ρ_{min}/w = 0.200 and 0.400. Next, Table 9 presents the optimal shapes for a hole aspect ratio of h:w ≈ 2:1 and minimum radius of curvature constraints of ρ_{min}/w = 0.222 and 0.430. Finally, Table 10 presents the optimal shapes for a hole aspect ratio of h:w ≈ 3:1 and minimum radius of curvature constraints of ρ_{min}/w = 0.300 and 0.570.

4.2 Large 2D plate with inclined slotted hole constrained by inclined line

The general geometry of a centrally-located inclined slotted hole (a slot with semicircular ends) in a large plate is depicted in Figure 13a, together with the constraint conditions. The radius of each of the two semicircular ends is r, the distance between the centres of the two semicircles is b, and the angle of inclination of the hole from the vertical is θ. For this example, the slotted hole is inclined at an angle θ = 16° to the vertical, and a fixed geometric constraint line also inclined at θ = 16° is included. The latter is considered to be a very significant geometric constraint, as it applies over a large portion of the hole boundary. A uniform uniaxial in-plane tension load S_2 = 100 MPa was applied to the plate in the y-direction. When b/r = 0, the initial hole has a circular shape and the inclined constraint line is tangent to this circle at one point on its circumference. A range of initial holes with values of b/r = 0, 2 and 4 were optimised with two different values of minimum radius of curvature constraint, ρ_{min}/u_T = 0.02 and 0.04. These two values of ρ_{min}/u_T represent typical minimum radii of curvature that are of practical interest when reworking cracked aircraft components. The local detail of the FEA mesh for a typical optimal shape is shown in Figure 13b for the case where b/r = 2 and ρ_{min}/u_T = 0.02.

Table 11 presents the results of shape optimisations carried out using the above range of b/r and ρ_{min}/u_T values. There are n = 4 stress peaks around the hole boundary, denoted by stress concentration factors $\{K_{tj}\}_{j=1}^{4}$. The table also gives the normalised lengths $\{\mu_j\}_{j=1}^{4}$ of each distinct flat-topped stress zone where the stress is within 1% of the peak value in each subregion, together with the total normalised length of the hole boundary meeting this criterion. The K_t values for the initial starting shapes are also provided for ease of comparison. As in the previous example, the aspect ratios of the optimal shapes hardly change from that of the initial shapes, even though there are substantial stress reductions; what changes is the area of the hole. As expected, for a given hole aspect ratio and loading case, we find that as $\rho_{min}/u_T \to 0$ then $\mu_T \to 1$. Because this case is more highly constrained than the previous example, the values of μ_T that are obtained are somewhat lower. It is noted once again that the only points on the boundary that are not in a constant stress zone are those that have locally invoked the ρ_{min} constraint or the inclined line constraint (as shown in Figure 12).

When b/r = 2, the hole shape was representative of an F-111 fuel flow vent hole, which had previously been the subject of single-peak shape optimisation (Heller *et al.* 1999), providing a

useful point of comparison with the results from the present multi-peak method. Values of $c = 4r = 50$ mm and $s = 0.0045$ were used for the optimisation in accordance with Equation (1). The effects of these two types of constraints on the optimal shapes can be seen in Figure 14a, which also shows the starting location for the measurement of the arc length u around the hole boundary, at $(x, y) = (b\cos 16°, b\sin 16°)$. The variation of the stress concentration factor K_t around the hole boundary as a function of normalised arc length μ for the initial slotted hole and the final free-form optimal holes is shown in Figure 14b. Here we define $K_t = \sigma_i^j / S_2$. Note that the hole boundary is convex in regions of compressive stress and concave where the stresses are tensile. Between the $n = 4$ zero crossings there are the same number of major stress peaks, which from left to right are $\{K_{tj}\}_{j=1}^4 = \{3.288, -1.045, 3.254, -1.051\}$. For the optimal shape with $\rho_{min}/u_T = 0.02$, these peaks have been reduced to flat-topped zones with values of $\{K_{tj}\}_{j=1}^4 = \{2.558, -0.820, 1.847, -0.727\}$, which are uniform to within a tolerance of 0.001. This is a reduction in each of the respective peak stresses of 22.2%, 21.5%, 43.2% and 30.8%. The high degree of stress uniformity indicates excellent convergence in the optimal solution, and 66% of the circumference has regions of uniform stress ($\mu_T = 0.661$). Because of the action of the constraint line, the extent of the uniform stress zone between $0.097 \le \mu \le 0.190$ ($\mu_1 = 0.093$) is much less than that between $0.394 \le \mu \le 0.716$ ($\mu_3 = 0.322$). Compared to a reduction of only 11% in the maximum absolute peak stress obtained using the prior single-peak algorithm (Heller *et al.* 1999), the multi-peak method has produced a much greater reduction of 23%.

5. Extension to 3D shape optimisation

The multi-peak shape optimisation method has also been applied to problems where the nodal displacements are computed using the full 3D stress data around the boundary of the hole. The particular implementation that has been chosen tackles ease of manufacturing considerations by maintaining a constant profile for the hole through the thickness of the structure. In this way, the shape optimisation is in fact quasi-3D in nature with respect to the calculated geometry. We look at the stress variation through the thickness of the component along lines of varying z for any given constant x and y at nodal points on the boundary of the hole, taking into account the full 3D effects on the stress distribution. The maximum value of the through-thickness stress is then used for each of the individual σ_i^j when the nodal movement is calculated. Hence, Equation (1) is modified so that, at any angular position i around the boundary corresponding to a stress zone j, we can calculate

$$d_i^j = \left(\frac{\sigma_i^j - \sigma_{th}^j}{\sigma_{th}^j} \right) sc\,, \quad j = 1, 2, \ldots, n$$

$$\sigma_i^j = \max(\sigma_{i,k}^j) \text{ if } \sigma_i^j > 0 \text{ or } \sigma_i^j = \min(\sigma_{i,k}^j) \text{ if } \sigma_i^j < 0\,, \quad k = 1, 2, \ldots, p \tag{5}$$

$$\sigma_{th}^j = \max(\sigma_i^j) \text{ if } \sigma_i^j > 0 \text{ or } \sigma_{th}^j = \min(\sigma_i^j) \text{ if } \sigma_i^j < 0$$

where $\sigma_{i,k}^j$ is the stress at the k-th through-thickness location at the i-th angular position around the stress concentrator, and p is the number of through-thickness nodal points (and p is constant around the boundary of the stress concentrator).

By making the in-plane x and y components of the computed nodal movement identical along a through-thickness line passing through the node and parallel to the z-axis, the shape can be manufactured using simple machining operations, as the profile remains the same through the entire thickness of the component.

5.1 Central hole in thick 3D plate under uniaxial loading

In this example, we consider a centrally-located elliptical hole in a large plate of thickness t subjected to a uniaxial in-plane loading. This is analogous to the hole in a 2D plate that was analysed previously in Section 3.1. The geometry is the same as that shown in Figure 4, but with the addition of the plate thickness parameter. In the FEA, the height and width of the plate were H = 1000 mm and W = 1000 mm, and the plate thickness was t = 90 mm. Two different hole aspect ratios were studied. The first corresponded to an initial circular hole with h:w = 1:1 (h = w = 50 mm). The second corresponded to an initial elliptical hole with h:w = 2:1 (h = 2w = 100 mm). The applied uniaxial stress was S_2 = 100 MPa for both cases. Values of c = 100 mm and s = 0.005 were used for the shape optimisation. For the circular holes, the imposed minimum radius of curvature constraints consisted of cases with ρ_{min}/u_T = 0.020, 0.040, and 0.084. For the elliptical holes, the imposed minimum radius of curvature constraints consisted of cases with ρ_{min}/u_T = 0.020 and 0.040. Apart from the minimum radius of curvature constraints, the shape was unconstrained during the optimisation process. The effects of the finite plate size are expected to be small, as the dimensions of the plate are large in comparison to the hole dimensions ($H/h \geq 5$).

Figure 15 shows the local detail of the ⅛-symmetry FEA mesh for a typical 2:1 optimal hole with ρ_{min}/u_T = 0.020. Note that the mesh has been refined near the plate surface to improve the accuracy of the stress calculations in this region (as the stress gradient is higher here than at the midplane of the plate). Figure 16a shows a comparison of the shape of the initial 2:1 elliptical hole with the resulting optimal holes for ρ_{min}/u_T = 0.020 and 0.040. Although the optimal holes are somewhat larger in area, they both retain an approximately 2:1 aspect ratio. Figure 16b shows the variation of the peak through-thickness K_t around the hole boundary for the initial elliptical hole and the free-form optimal hole that was obtained for ρ_{min}/u_T = 0.020 and 0.040. Between the n = 4 zero crossings, the elliptical hole has four major stress peaks, which from left to right are $\{K_{tj}\}_{j=1}^{4}$ = {1.941, –1.153, 1.941, –1.153}, and the peak tensile stress concentration factor occurs at the two points located at (x, y, z) = (±50, 0, 0) on the midplane of the plate. For the optimal shape with ρ_{min}/u_T = 0.020, these stress peaks have been reduced to flat-topped zones of constant stress with values of $\{K_{tj}\}_{j=1}^{4}$ = {1.648, –0.802, 1.648, –0.802}, which are uniform to within a tolerance of 0.002. This corresponds to reductions in the peak stresses of 15.1% for the tensile peaks, and 30.4% for the compressive peaks. Just as in earlier examples, the high degree of stress uniformity that has been achieved indicates excellent convergence in the optimal solution. The extents of the uniform regions of tensile and compressive stress were $\{\mu_j\}_{j=1}^{4}$ = {0.363, 0.188, 0.263, 0.188}, with a combined μ_T = 0.925.

Table 12 presents the results of the 3D multi-peak shape optimisation of the initial circular holes for ρ_{min}/u_T = 0.020, 0.040, and 0.084, as well as the results for the initial 2:1 elliptical holes for ρ_{min}/u_T = 0.02 and 0.04. For ease of comparison, K_t values for the initial starting shapes are also provided. It is evident that, where the aspect ratios of the optimal holes have

increased from that of the starting configuration, the increase is negligible. It is also clear the lowest peak stresses and greatest total length of uniform stress around the boundary are obtained for the smaller value of ρ_{min}/u_T. As expected, the holes with larger aspect ratio produce the lowest peak stresses. It is noted once again that the only points on the boundary that are not in a constant stress zone (based on the maximum through-thickness stress at each angular location on the hole boundary) are those that have locally invoked the ρ_{min} constraint (as shown in Figure 14).

Focussing now on the case involving the initial circular hole, which has $n = 4$ zero crossings occurring between the four major stress peaks around its boundary, Table 12 lists the initial stress peaks as being $\{K_{tj}\}_{j=1}^{4}$ = {3.074, –1.118, 3.074, –1.118}. The resulting optimal hole for this case with ρ_{min}/u_T = 0.020 was of approximately 1:1 aspect ratio, and it produced reductions in the peak stresses of 26.8% for the tensile peaks ($K_{t1} = K_{t3}$ = 2.252) and 19.6% for the compressive peaks ($K_{t2} = K_{t4}$ = –0.899). Once again, a high degree of stress uniformity in the flat-topped zones was achieved, and 90% of the circumference of the optimal hole has zones of uniform stress (μ_T = 0.900). In comparison, the results for ρ_{min}/u_T = 0.040 produced peak values of tensile and compressive K_t that were greater than for the ρ_{min}/u_T = 0.020 case, in this case by 2.8% and 0.7%, respectively. This was as expected, as was the fact that the total length of the zones of uniform stress was also less, by 8.3% in this case. Similar trends were also evident with the 2:1 holes.

The positive-K_t value obtained for the 1:1 aspect ratio hole with ρ_{min}/u_T = 0.020 for the 3D case presented in Table 12 is 2.7% higher than the corresponding 2D results presented in Table 1. On the other hand, the positive-K_t value obtained for the 2:1 aspect ratio hole with ρ_{min}/u_T = 0.020 for the 3D case is 7.5% lower than the corresponding 2D results. A plot comparing the 2D and 3D optimal shapes obtained for the 2:1 aspect ratio hole is presented in Figure 17a. It is seen that the two shapes are very similar. A close-up view of the 2D and 3D shapes in the corner region is shown in Figure 17a, where it is apparent that the 2D shape has removed a little more material. The comparison of the results obtained using 2D and 3D plates shows that, in order to achieve the best possible results, it is important to account for the 3D nature of the stress distribution at a stress concentrator boundary.

6. Guidelines for performing multi-peak shape optimisation

Some important and useful guidelines are given below regarding the convergence of a solution during multi-peak shape optimisation, and also the selection of a good initial starting shape.

6.1 Monitoring solution convergence

From a detailed examination of the examples in Sections 4.1 and 4.2, it was found that monitoring the individual lengths of the constant stress regions, as well as their total sum, was very useful in determining the point at which no further iterations were required. For example, two typical cases are given, the first in Figure 18 for a 2:1 hole close to an edge, and the second in Figure 19 for a 2:1 hole geometrically constrained by an inclined line, both for ρ_{min}/u_T = 0.02. The graphs show that the length of each constant stress region increases

steadily with increasing iteration number. Eventually a very distinct point is reached where the individual lengths of the constant stress regions no longer increases. The graphs also show the total combined normalised length of the constant stress zones around the entire boundary of the hole, μ_T, as computed from Equation (2). Once all the individual lengths of the constant stress zones have become constant, μ_T also plateaus. There is no significant further improvement in the stresses for iterations beyond this point. Hence, final plateauing of μ_T is considered to be a good indicator of solution convergence. As expected, for geometrically unconstrained holes for a given hole aspect ratio and loading case, we find that as $\rho_{min}/u_T \to 0$ then $\mu_T \to 1$, and for $\rho_{min}/u_T = 0$ then $\mu_T = 1$. For constrained holes, the same trend applies, but it is not possible to achieve the limit of $\mu_T = 1$ even when $\rho_{min}/u_T = 0$.

For all cases that were investigated, it is noted that the only points on the boundary that were not in a constant stress zone were those that had locally invoked geometric constraints involving the ρ_{min} constraint or the inclined line constraint.

6.2 Selection of starting shape to obtain optimal shape

All the analyses undertaken thus far indicate that the "optimal" shapes are all dependent on the aspect ratio of the initial starting shape. The numerically determined optimal shapes effectively have the same aspect ratio as the starting shape, but they have increased area and reduced stress. Hence, in practice, analysts should undertake numerical shape optimisation using shapes with a few different aspect ratios, typically in the range 1:1 to 3:1, and then select the solution that best suits their requirements. Note that increasing the aspect ratio provides progressively lower peak stresses.

To efficiently determine an optimal shape with a minimum of iterations, it is desirable to: (i) align the initial hole with the direction of the dominant loading, and (ii) use a smooth starting shape (such as an ellipse). The present algorithm has been well validated to apply for the case where the minimum radius of curvature of any point on the initial shape is greater than the user-selected minimum radius of curvature constraint, ρ_{min}. Hence, this fact needs to be taken into account when selecting the initial starting shape.

6.3 Alignment of optimal shapes with bulk principal stress directions

As expected, the optimal shapes obtained here for reducing stress concentrations display the characteristic feature that their boundaries tend to align themselves with the bulk principal stress directions. This is certainly very evident for the optimal holes in the uniaxially-loaded plates considered in this report, and it is the same for optimal fillets in tension and bending (Waldman *et al*. 2001). Further examples of this characteristic can be found in the study of structural loadflow around various optimal shapes (Waldman *et al*. 2002b). For the case of an optimal elliptical hole in a biaxially-loaded plate, where the hole displays constant stress around its entire boundary, the ratio of the minor and major axes of the ellipse is equal to the ratio of the applied bulk principal stresses. It is also well known that the major axis of the optimal ellipse will be aligned with the dominant bulk principal stress direction. These optimality conditions related to an elliptical hole in a biaxial stress field were long ago identified by Durelli and Murray (1943). Hence, when choosing an initial shape from which

to start the optimisation process, it is useful to try and align the major axis of the shape with the dominant bulk principal stress direction.

7. Modification of sharp-cornered optimal hole shapes to improve robustness

Although optimal shapes with sharp corners are the theoretical ideal optimal solutions for the loading conditions considered here, in practice it is usually desirable to modify these shapes by removing the sharp re-entrant corners. This serves to enhance the robustness of the designs under moderately different loadings or situations where loading misalignment away from the design conditions can occur. An additional important benefit is that the smoothed optimal shape will be easier to manufacture as it does not contain a re-entrant corner. In prior work, the design of robust shapes has been accomplished by perturbing the loads (Heller *et al*. 2009).

As an example, Figure 20 shows the normalised contours of maximum principal stress for a typical ideal sharp-cornered optimal hole geometry taken from Burchill and Heller (2004b). Looking at the bottom right corner of the hole, it is apparent that the material that is located above and to the right of the sharp corner is very lowly stressed. A similar situation exists for the other three corners as well. As a result, it is possible to remove this redundant material without significantly increasing the value of the stress concentration factor relative to the original sharp-cornered optimal design.

7.1 Method for adding an undercut to ideal optimal holes

One method of designing robust rounded shapes for both sharp-cornered ideal optimal fillet shapes and optimal hole shapes is to create intentional undercuts in the re-entrant corner regions of the optimal hole profile. Such an undercut is depicted by the curve between Points A and B in Figure 21. This design approach is relatively easy to apply, and it also has the benefit of removing a minimal amount of redundant material from the low-stressed region. It is generally convenient to use a circular undercut, although other smooth shapes could also feasibly be utilised in a similar manner. Note that the method that is being proposed here can also be used with optimal fillets, which also feature sharp corners in their designs (see Waldman, Heller and Chen 2001).

A schematic for a robust optimal hole design with a circular undercut is shown in Figure 22. It is possible to calculate an estimate of the local radius of curvature of the original optimal profile at what used to be the re-entrant corner (Point P_3) simply by fitting a circle to the three points P_1, P_2, and P_3. A circular arc with this radius has then been used to create the undercut. Here it is necessary to ensure that the circular undercut is positioned in such a way that continuity of slope between the undercut and the original optimal profile is enforced at Point P_3, which is the transition point between the two curves. This requirement is readily met by using the centre of the previously determined circle as the centre of the circular arc undercut.

Consider a circle of radius r whose centre is located at point $P_c = (x_c, y_c)$. The equation of this circle is:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{6}$$

If $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, and $P_3 = (x_3, y_3)$ are three points that lie on a circle, the radius, r, and the coordinates of the centre of the circle, (x_c, y_c), can be computed using the following set of three equations:

$$x_c = (ed - cf)/g \tag{7}$$

$$y_c = (af - eb)/g \tag{8}$$

$$r = \sqrt{(x_c - x_3)^2 + (y_c - y_3)^2} \tag{9}$$

where

$$a = 2(x_2 - x_1)$$

$$b = 2(x_3 - x_2)$$

$$c = 2(y_2 - y_1)$$

$$d = 2(y_3 - y_2)$$

$$e = {x_2}^2 + {y_2}^2 - {x_1}^2 - {y_1}^2$$

$$f = {x_3}^2 + {y_3}^2 - {x_2}^2 - {y_2}^2$$

$$g = ad - bc$$

If the value of g is zero, then the three points are collinear and the equations given above no longer apply.

The location of the point $P_f = (x_f, y_f)$ in Figure 22, which is where the upper end of the circular arc of the undercut meets with the flat boundary of the hole or fillet, can be computed as follows:

$$x_f = x_3 \tag{10}$$

$$y_f = 2y_c - y_3 \tag{11}$$

The depth of the undercut is then given by:

$$\Delta l = x_c + r - x_3 \tag{12}$$

7.2 Example of an undercut optimal hole shape

The example optimal hole considered here is for the case of uniaxial loading (S_2:S_1 = 1:0) and a single hole of aspect ratio w/h = 1 (see Figure 4 for a schematic diagram of the geometry and loading). The desired design half-length of the hole is h = 100 mm. The sharp-cornered optimal profile has K_t = 2.147, which is 29 per cent less than an equivalent circular hole in the same finite plate. The last three points in the optimal hole profile leading up to and including

the re-entrant corner are $P_1 = (x_1, y_1) = (98.089, 18.196)$, $P_2 = (x_2, y_2) = (99.085, 19.024)$, and $P_3 = (x_3, y_3) = (100, 19.996)$.

Using Equations (6)–(9), the equation of the circle passing through those three points is:

$$(x - 91.7067)^2 + (y - 26.863)^2 = 10.7821^2$$

The depth of the resulting undercut is $\Delta l = 2.489$ mm.

Finite element models of the two hole geometries were created and analysed, one with the re-entrant corner and the other with the robust circular undercuts added. The chosen size of the hole was 1/20 the size of the plate in order to minimise finite-width plate effects (see Burchill and Heller, 2004b). The contours of maximum principal stress for the two geometries are plotted in Figure 20 and Figure 23, and these correspond to an applied uniaxial remote tensile stress of $S_1 = 100$ MPa. The geometry with the undercut has a $K_t = 2.156$, which is only 0.4 per cent greater than for the sharp-cornered optimal profile, thus confirming the usefulness of this approach.

8. Conclusion

For the first time, a practical fully-automated shape optimisation capability has been demonstrated for minimising multiple stress peaks in stress concentrators that are subject to considerable geometric and minimum radius of curvature constraints. The FORTRAN 90 source code for the shape optimisation program is provided in Appendix A. The DSTO multi-peak shape optimisation method represents a new standard in general-purpose shape optimisation technology, computing shapes with superior stress reducing properties compared to other methods. It is very useful for reducing stresses so as to help maximise the fatigue life of a wide range of engineering structures with stress concentrations, with particular applicability to free-form shape optimisation of both holes and fillets. Compared to the single-peak method of shape optimisation, the multi-peak approach is able to reduce the size of the any compressive stress peaks at the same time as it reduces the tensile stress peaks. K_t values and tables of transferable optimal shape coordinates are provided for selected classic benchmarks and other cases of significance for holes of various aspect ratios in large plates under uniaxial or biaxial remote loading. The new multi-peak method is especially applicable to creating individual optimal shapes for the unique geometry and loading cases that are typical of actual practical problems. It also embodies an easily-computed parameter for evaluating the optimality of any proposed optimal shape. This parameter consists of the summation of the lengths of the flat-topped stress zones that exist around the perimeter of the stress concentrator whose boundary shape has been optimised. The effectiveness of a method for modifying sharp-cornered optimal shapes to improve their robustness, by introducing a circular undercut into their design, has also been demonstrated and shown to be both simple and effective in its application.

9. References

Baud RV, 1934: Fillet profiles for constant stress. *Product Engineering*, April, pp 133–134.

Burchill M, Heller M, 2004a: Optimal notch shapes for loaded plates. *Journal of Strain Analysis for Engineering Design*, Vol 39, No 1, pp 99–116.

Burchill M, Heller M, 2004b: Optimal free-form shapes for holes in flat plates under uniaxial and biaxial loading. *Journal of Strain Analysis for Engineering Design*, Vol 39, No 6, pp 595–614.

Cherepanov GP, 1974: Inverse problems of the plane theory of elasticity. *Journal of Applied Mathematics and Mechanics*, Vol 38, No 6, pp 915–930 (English translation of Russian *Prik Matem Mekhan*, Vol 38, No 6, 1974, pp 963–979). See correction to Equation 3.17 in Vigdergauz (1976).

Cline AK, 1974a: Scalar- and planar-valued curve fitting using splines under tension. *Communications of the ACM*, Vol 17, No 4, April, pp 218–220.

Cline AK, 1974b: Algorithm 476–Six subprograms for curve fitting using splines under tension. *Communications of the ACM*, Vol 17, No 4, April, pp 220–223.

Dongarra JJ, Grosse E, 1987: Distribution of mathematical software via electronic mail. *Communications of the ACM*, Vol 30, No 5, May, pp 403–407.

Dhir SK, 1981: Optimization of a class of hole shapes in plate structures. *Journal of Applied Mechanics*, Vol 48, December, pp 905–908.

Durelli AJ, Brown K, Yee P, 1978: Optimization of geometric discontinuities in stress fields. *Experimental Mechanics*, Vol 18, August, pp 303–308.

Durelli AJ, Murray WM, 1943: Stress distribution around an elliptical discontinuity in any two-dimensional, uniform and axial, system of combined stress. *Proceedings of the Society for Experimental Stress Analysis*, Vol 1, No 1, pp 19–31.

Durelli AJ, Rajaiah K, 1979: Optimum hole shapes in finite plates under uniaxial load. *Journal of Applied Mechanics*, Vol 46, September, pp 691–695.

Durelli AJ, Rajaiah K, 1980: Lighter and stronger. *Experimental Mechanics*, Vol 20, No 11, November, pp 369–380.

Durelli AJ, Rajaiah K, 1981: Quasi-square hole with optimum shape in an infinite plate subjected to in-plane loading. *Transactions of the ASME, Journal of Mechanical Design*, Vol 103, October, pp 866–870.

Heller SR, 1969: Stress concentration factors for a rectangular opening with rounded corners in a biaxially loaded plate. *Journal of Ship Research*, September, pp 178–184.

Heller M, Kaye R, Rose LRF, 1999: A gradientless procedure for shape optimisation. *Journal of Strain Analysis for Engineering Design*, Vol 34, No 5, pp 323–336.

Heller M, McDonald M, Burchill M, Watters KC, 2002: F-111 airframe life extension through rework shape optimisation of critical features in the wing pivot fitting. In: Proceedings of the 6th Joint FAA/DoD/NASA Aging Aircraft Conference, 16–19 September, San Francisco, USA.

Heller M, Burchill M, Wescott R, Waldman W, Kaye R, Evans R, McDonald M, 2009: Airframe life extension by optimised shape reworking – overview of DSTO developments. In: 'ICAF 2009, Bridging the Gap between Theory and Operational Practice', Proceedings of the 25th Symposium of the International Committee on Aeronautical Fatigue, MJ Bos (ed), 27–29 May, Rotterdam, The Netherlands.

Heywood RB, 1969: Photoelasticity for Designers (Chapter 11: Improvement of Designs), Pergamon Press.

Lansard R, 1955: Fillets without stress concentrations. *Proceedings of the Society for Experimental Stress Analysis*, Vol 13, No 1, pp 97–104.

Mattheck C, Burkhardt S, 1990: A new method of structural shape optimisation based on biological growth. *International Journal of Fatigue*, Vol 12, No 3, pp 185–190.

McDonald M, Heller M, 2004: Robust shape optimization of notches for fatigue-life extension. *Structural and Multidisciplinary Optimization*, Vol 28, No 1, pp 55–68.

Morris AH, Jr, 1993: NSWC Library of Mathematics Subroutines. NSWCDD/TR-92/425, Naval Surface Warfare Center, Dahlgren Division, Dahlgren, Virginia, USA, January.

Neuber H, 1972: Zur optimirung der spannungskonzentration. Continuum Mechanics and Related Problems of Analysis, Nauka Publishing House, Moscow, pp 375–380.

Rajaiah K, Naik NK, 1983: Hole shapes with minimum stress concentration in infinite isotropic plates using conformal transformation. *ISME Journal of Engineering Design*, Vol 1, No 1, April, pp 15–19.

Schnack E, 1983: Computer-simulation of an experimental method for notch-shape-optimization. In: 'Simulation in Engineering Sciences,' Proceedings of the International Symposium, J Burger and Y Jarny (eds), Nantes, May, pp 311–316.

Vigdergauz SB, 1976: Integral equation of the inverse problem of the plane theory of elasticity. *Journal of Applied Mathematics and Mechanics*, Vol 40, No 3, pp 518–522 (English translation of Russian *Prik Matem Mekhan*, Vol 40, No 3, 1976, pp 566–569). Includes correction to Equation 3.17 in Cherepanov (1974).

Vigdergauz SB, Cherkayev AV, 1986: A hole in plate, optimal for its biaxial extension-compression. *Journal of Applied Mathematics and Mechanics*, Vol 50, No 3, pp 401–404 (*Prikl Matem Mekhan*, Vol 50, No 3, pp 524–528, 1986).

Vigdergauz S, 2006: The stress-minimizing hole in an elastic plate under remote shear. *Journal of Mechanics of Materials and Structures*, Vol 1, No 2, pp 389–406.

Vigdergauz S, 2008: Shape optimization in an elastic plate under remote shear: from single to interacting holes. *Journal of Mechanics of Materials and Structures*, Vol 3, No 7, pp 1341–1363.

Vigdergauz S, 2010: Energy-minimizing openings around a fixed hole in an elastic plate. *Journal of Mechanics of Materials and Structures*, Vol 5, No 4, pp 661–677.

Waldman W, Heller M, Chen G, 2001: Optimal free-form fillet shapes for tension and bending. *International Journal of Fatigue*, Vol 23, pp 509–523.

Waldman W, Heller M, McDonald M, Chen G, 2002a: Developments in rework shape optimisation for life extension of aging airframes. In: 'Applied Mechanics: Progress and Applications', Proceedings of the 3rd Australasian Congress on Applied Mechanics, edited by L Zhang, L Tong, J Gal, 20–22 February, Sydney, Australia, pp 695–702.

Waldman W, Heller M, Kaye R, Rose LRF, 2002b: Advances in structural loadflow visualisation. *Engineering Computations*, Vol 19, No 3, pp 305–326

Waldman W, Heller M, Rose LRF, 2003: Shape optimisation of two closely-spaced holes for fatigue life extension. DSTO-RR-0253, Air Vehicles Division, Defence Science and Technology Organisation, Australia.

Waldman W, Heller M, 2005: Shape optimisation of holes for multi-peak stress minimisation. In: 'Advances in Applied Mechanics,' Proceedings of the 4th Australasian Congress on Applied Mechanics, edited by YM Xie, AP Mouritz, AA Khatibi, C Gardiner, WK Chiu, 16–18 February, Melbourne, Australia, pp 189–195.

Wheeler L, 1976: On the role of constant-stress surfaces in the problem of minimizing elastic stress concentration. *International Journal of Solids and Structures*, Vol 12, Issue 11, pp 779–789.

This page is intentionally blank

Table 1: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large plate with a central hole subject to uniaxial remote loading (S_2:S_1 = 1:0).*

Method	Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1} (=K_{t3})	K_{t2} (=K_{t4})	μ_1 (= μ_3)	μ_2 (= μ_4)	$\sum_{j=1}^{4}\mu_j$
D & R[1]	1.000	1.000	0.066	—	2.542	–0.869	0.218	0.109	0.655
R & N[2]	—	1.000	0.074	0.249	2.468	–0.912	0.050	0.029	0.158
Dhir[3]	—	1.000	≈0.15	—	2.470	–0.920	—	—	—
Schnack[4]	1.000	1.000	≈0	≈0	2.326	–0.804	0.276	0.000	0.552
Single-peak	1.000	1.046	0.01999	0.148	2.191	–0.890	0.263	0.050	0.625
Single-peak	1.000	1.020	0.04012	0.288	2.260	–0.881	0.250	0.038	0.575
Multi-peak	1.000	1.056	—	0.100	2.172	–0.837	0.275	0.188	0.925
Multi-peak	1.000	1.046	0.02024	0.150	2.193	–0.829	0.263	0.188	0.900
Multi-peak	1.000	1.031	—	0.200	2.219	–0.833	0.263	0.175	0.875
Multi-peak	1.000	1.020	0.03982	0.286	2.259	–0.836	0.250	0.163	0.825
Multi-peak	1.000	1.019	—	0.300	2.265	–0.836	0.250	0.163	0.825
Multi-peak	1.000	1.009	—	0.400	2.320	–0.844	0.225	0.150	0.750
Multi-peak	1.000	1.002	—	0.500	2.381	–0.857	0.213	0.125	0.675
R & N[2]	—	2.000	0.078	0.387	1.862	–0.954	0.125	0.000	0.250
Single-peak	2.000	2.019	0.02011	0.218	1.782	–0.881	0.350	0.025	0.750
Single-peak	2.000	2.000	0.04005	0.422	1.816	–0.907	0.338	0.000	0.675
Multi-peak	2.000	2.070	—	0.100	1.761	–0.809	0.363	0.113	0.950
Multi-peak	2.000	2.020	—	0.200	1.780	–0.837	0.363	0.100	0.925
Multi-peak	2.000	2.019	0.02011	0.218	1.782	–0.837	0.350	0.100	0.900
Multi-peak	2.000	2.007	—	0.300	1.798	–0.848	0.363	0.088	0.900
Multi-peak	2.000	2.001	—	0.400	1.813	–0.863	0.338	0.075	0.825
Multi-peak	2.000	2.001	0.04008	0.422	1.817	–0.867	0.338	0.075	0.825
Multi-peak	2.000	2.000	—	0.500	1.828	–0.885	0.338	0.063	0.800

[1] D & R = Durelli and Rajaiah (1981)
[2] R & N = Rajaiah and Naik (1983)
[3] Dhir = Dhir (1981)
[4] Schnack = Schnack (1983)

Table 2: *Coordinates of a ¼-symmetric centrally-located optimal hole of different aspect ratios ($h/w \approx 1.0$ and $h/w \approx 2.0$) and a variety of normalised minimum radius of curvatures (ρ_{min}/w) in a uniaxially-loaded large plate (S_2:S_1 = 1:0) , optimised using multi-peak method.*

h/w = 1.046 ρ_{min}/w = 0.150 ρ_{min}/u_T = 0.02024 K_t = 2.193		h/w = 1.020 ρ_{min}/w = 0.286 ρ_{min}/u_T = 0.03982 K_t = 2.259		h/w = 2.019 ρ_{min}/w = 0.218 ρ_{min}/u_T = 0.02011 K_t = 1.782		h/w = 2.001 ρ_{min}/w = 0.422 ρ_{min}/u_T = 0.04005 K_t = 1.817	
x/w	y/w	x/w	y/w	x/w	y/w	x/w	y/w
1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.99971	0.04632	0.99969	0.04489	0.99976	0.06776	0.99975	0.06580
0.99884	0.09263	0.99877	0.08977	0.99902	0.13551	0.99898	0.13161
0.99739	0.13892	0.99723	0.13463	0.99780	0.20326	0.99771	0.19740
0.99534	0.18519	0.99507	0.17947	0.99608	0.27100	0.99592	0.26318
0.99270	0.23143	0.99227	0.22427	0.99387	0.33872	0.99361	0.32894
0.98945	0.27764	0.98882	0.26903	0.99115	0.40642	0.99077	0.39469
0.98558	0.32379	0.98470	0.31373	0.98792	0.47411	0.98739	0.46041
0.98106	0.36989	0.97990	0.35836	0.98417	0.54176	0.98348	0.52610
0.97588	0.41592	0.97438	0.40291	0.97989	0.60938	0.97900	0.59175
0.97001	0.46186	0.96811	0.44736	0.97506	0.67697	0.97395	0.65736
0.96342	0.50771	0.96106	0.49169	0.96968	0.74452	0.96831	0.72292
0.95606	0.55344	0.95317	0.53588	0.96372	0.81201	0.96207	0.78843
0.94790	0.59903	0.94438	0.57990	0.95716	0.87945	0.95519	0.85388
0.93886	0.64445	0.93460	0.62372	0.94999	0.94683	0.94765	0.91925
0.92888	0.68968	0.92374	0.66727	0.94217	1.01414	0.93942	0.98454
0.91783	0.73466	0.91165	0.71050	0.93367	1.08136	0.93045	1.04973
0.90562	0.77934	0.89815	0.75332	0.92446	1.14849	0.92072	1.11481
0.89201	0.82362	0.88295	0.79555	0.91449	1.21551	0.91015	1.17976
0.87680	0.86736	0.86558	0.83695	0.90371	1.28241	0.89870	1.24456
0.85954	0.91034	0.84573	0.87721	0.89204	1.34915	0.88626	1.30918
0.83946	0.95209	0.82034	0.91423	0.87943	1.41573	0.87275	1.37358
0.81577	0.99188	0.78947	0.94682	0.86575	1.48209	0.85802	1.43772
0.78106	1.02255	0.75388	0.97418	0.85091	1.54821	0.84194	1.50153
0.73865	1.04117	0.71445	0.99563	0.83468	1.61399	0.82418	1.56489
0.69258	1.04597	0.67214	1.01065	0.81693	1.67938	0.80452	1.62769
0.64701	1.03768	0.62801	1.01886	0.79709	1.74417	0.78217	1.68958
0.60086	1.03383	0.58314	1.02007	0.77504	1.80824	0.75644	1.75015
0.55484	1.02851	0.53828	1.01838	0.74861	1.87063	0.72477	1.80784
0.50875	1.02396	0.49344	1.01625	0.71730	1.93072	0.68453	1.85990
0.46263	1.01966	0.44862	1.01371	0.66905	1.97830	0.63669	1.90508
0.41647	1.01588	0.40380	1.01126	0.60853	2.00876	0.58240	1.94226
0.37027	1.01254	0.35897	1.00895	0.54158	2.01917	0.52299	1.97056
0.32405	1.00964	0.31413	1.00688	0.47389	2.01619	0.45991	1.98928
0.27780	1.00716	0.26928	1.00506	0.40623	2.01277	0.39469	1.99797
0.23153	1.00509	0.22441	1.00352	0.33857	2.00930	0.32893	2.00021
0.18524	1.00342	0.17954	1.00225	0.27088	2.00634	0.26314	2.00069
0.13894	1.00213	0.13466	1.00127	0.20318	2.00400	0.19735	2.00059
0.09263	1.00121	0.08978	1.00057	0.13546	2.00232	0.13156	2.00031
0.04632	1.00066	0.04489	1.00014	0.06773	2.00131	0.06578	2.00009
0.00000	1.00048	0.00000	1.00000	0.00000	2.00097	0.00000	2.00001

Table 3: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large plate with a central hole undergoing reversed biaxial remote loading of S_2:S_1 = 1:–1.*

Method	Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1} (=K_{t3})	K_{t2} (=K_{t4})	μ_1 (= μ_3)	μ_2 (= μ_4)	$\sum_{j=1}^{4}\mu_j$
V & C[1]	–	1.000	0.000	–	2.732	–2.732	0.25*	0.25*	1.00*
D & R[2]	1.000	1.000	0.068	–	3.584	–3.584	0.040	0.040	0.160
R & N[3]	–	1.000	0.046	0.163	3.074	–3.074	0.077	0.077	0.308
Dhir[4]	–	1.000	–	–	3.070	–3.070	–	–	–
Multi-peak	1.000	1.000	0.000	0.000	2.844	–2.844	0.250	0.250	1.000
Multi-peak	1.000	1.000	–	0.100	2.912	–2.910	0.232	0.232	0.925
Multi-peak	1.000	1.000	0.02043	0.150	2.942	–2.940	0.225	0.225	0.900
Multi-peak	1.000	1.000	–	0.200	2.965	–2.940	0.225	0.225	0.900
Multi-peak	1.000	1.000	0.04001	0.286	3.021	–3.019	0.213	0.213	0.850
Multi-peak	1.000	1.000	–	0.300	3.034	–3.032	0.200	0.200	0.800
Multi-peak	1.000	1.000	–	0.400	3.107	–3.105	0.188	0.188	0.750
Multi-peak	1.000	1.000	–	0.500	3.187	–3.185	0.175	0.175	0.700

[1] V & C = Vigdergauz and Cherkayev (1986)
[2] D & R = Durelli and Rajaiah (1981)
[3] R & N = Rajaiah and Naik (1983)
[4] Dhir = Dhir (1981)
* Estimated value, not precisely specified

Table 4: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large plate with a central hole subject to reversed biaxial remote loading of S_2:S_1 = 1.377:–1.*

Method	Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1} (=K_{t3})	K_{t2} (=K_{t4})	μ_1 (= μ_3)	μ_2 (= μ_4)	$\sum_{j=1}^{4}\mu_j$
R & N[1]	–	2.000	0.02764	0.308	2.771	–2.867	0.013	0.012	0.050
Multi-peak	2.000	2.000	0.02002	0.222	2.424	–3.020	0.313	0.138	0.900
Multi-peak	2.000	2.000	0.03990	0.430	2.465	–3.199	0.300	0.113	0.825
R & N[1]	–	1.329	0.05155	0.314	2.686	–2.671	0.017	0.021	0.076
Multi-peak	1.329	1.329	0.01149	0.100	2.525	–2.525	0.273	0.191	0.926
Multi-peak	1.329	1.329	0.02034	0.174	2.561	–2.560	0.272	0.179	0.902
R & N[1]	–	1.000	0.05113	0.356	2.895	–2.442	0.040	0025	0.130
Multi-peak	1.000	1.000	0.01347	0.100	2.707	–2.329	0.212	0.225	0.874

[1] R & N = Rajaiah and Naik (1983)

Table 5: *Coordinates of a ¼-symmetric centrally-located optimal hole of aspect ratio $h/w = 1$ and a variety of normalised minimum radius of curvatures (ρ_{min}/w) in a large plate under remote reversed biaxial loading $S_2{:}S_1 = 1{:}{-1}$, optimised using multi-peak method.*

$h/w = 1$ $\rho_{min}/w = 0$ $\rho_{min}/u_T = 0$ $K_t = 2.844$		$h/w = 1$ $\rho_{min}/w = 0.150$ $\rho_{min}/u_T = 0.02043$ $K_t = 2.942$		$h/w = 1$ $\rho_{min}/w = 0.286$ $\rho_{min}/u_T = 0.04001$ $K_t = 3.021$	
x/w	y/w	x/w	y/w	x/w	y/w
1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.99991	0.04833	0.99988	0.04590	0.99985	0.04468
0.99964	0.09666	0.99950	0.09180	0.99939	0.08936
0.99918	0.14499	0.99888	0.13769	0.99863	0.13404
0.99854	0.19331	0.99800	0.18358	0.99755	0.17870
0.99772	0.24164	0.99686	0.22947	0.99614	0.22336
0.99671	0.28996	0.99546	0.27534	0.99440	0.26801
0.99551	0.33827	0.99378	0.32121	0.99231	0.31264
0.99411	0.38658	0.99180	0.36707	0.98985	0.35726
0.99251	0.43489	0.98952	0.41291	0.98698	0.40185
0.99072	0.48318	0.98691	0.45873	0.98367	0.44640
0.98871	0.53147	0.98394	0.50454	0.97987	0.49092
0.98648	0.57975	0.98057	0.55031	0.97550	0.53539
0.98403	0.62802	0.97676	0.59605	0.97047	0.57979
0.98133	0.67627	0.97239	0.64174	0.96456	0.62408
0.97837	0.72451	0.96737	0.68737	0.95765	0.66822
0.97515	0.77273	0.96148	0.73289	0.94884	0.71202
0.97156	0.82093	0.95809	0.75558	0.93831	0.75545
0.96773	0.86911	0.95431	0.77822	0.92114	0.79670
0.96515	0.91737	0.95009	0.80078	0.89776	0.83477
0.96569	0.96570	0.94514	0.82319	0.86873	0.86874
0.91736	0.96557	0.93948	0.84543	0.83477	0.89778
0.86910	0.96811	0.93049	0.86655	0.79670	0.92117
0.82092	0.97201	0.91839	0.88605	0.75545	0.93834
0.77273	0.97562	0.90346	0.90348	0.71203	0.94889
0.72451	0.97887	0.88604	0.91842	0.66822	0.95769
0.67627	0.98184	0.86655	0.93054	0.62408	0.96460
0.62802	0.98455	0.84544	0.93954	0.57979	0.97051
0.57975	0.98701	0.82320	0.94521	0.53539	0.97553
0.53147	0.98924	0.80079	0.95016	0.49093	0.97990
0.48318	0.99126	0.77823	0.95438	0.44641	0.98369
0.43488	0.99306	0.75559	0.95816	0.40185	0.98700
0.38658	0.99466	0.73289	0.96154	0.35726	0.98986
0.33827	0.99606	0.68737	0.96742	0.31264	0.99232
0.28996	0.99727	0.64175	0.97244	0.26801	0.99441
0.24164	0.99828	0.59605	0.97680	0.22336	0.99615
0.19331	0.99911	0.55031	0.98061	0.17870	0.99755
0.14499	0.99975	0.50454	0.98397	0.13404	0.99863
0.09666	1.00020	0.45874	0.98693	0.08936	0.99939
0.04833	1.00048	0.41291	0.98954	0.04468	0.99985
0.00000	1.00057	0.36707	0.99182	0.00000	1.00000
1.00000	0.00000	0.32121	0.99379	1.00000	0.00000
		0.27534	0.99547		
		0.22947	0.99687		
		0.18358	0.99801		
		0.13769	0.99888		
		0.09180	0.99950		
		0.04590	0.99988		
		0.00000	1.00000		

Table 6: *Coordinates of a ¼-symmetric centrally-located optimal hole of various aspect ratios (h/w) and a variety of normalised minimum radius of curvatures (ρ_{min}/w) in a large plate under remote reversed biaxial loading S_2:S_1 = 1.377:–1, optimised using multi-peak method.*

h/w = 2 ρ_{min}/w = 0.111 ρ_{min}/u_T = 0.02002 K_t = +2.424 and –3.020		h/w = 2 ρ_{min}/w = 0.215 ρ_{min}/u_T = 0.03990 K_t = +2.465 and –3.199		h/w = 1.329 ρ_{min}/w = 0.050 ρ_{min}/u_T = 0.01149 K_t = +2.525 and –2.525		h/w = 1.329 ρ_{min}/w = 0.087 ρ_{min}/u_T = 0.02034 K_t = +2.561 and –2.560	
x/w	y/w	x/w	y/w	x/w	y/w	x/w	y/w
1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
0.99991	0.06932	0.99989	0.06735	0.99986	0.06177	0.99984	0.06068
0.99962	0.13864	0.99955	0.13471	0.99942	0.12353	0.99934	0.12136
0.99915	0.20796	0.99897	0.20206	0.99869	0.18513	0.99852	0.18187
0.99848	0.27727	0.99817	0.26941	0.99766	0.24673	0.99736	0.24238
0.99762	0.34659	0.99714	0.33676	0.99634	0.30799	0.99587	0.30256
0.99657	0.41590	0.99586	0.40410	0.99472	0.36925	0.99403	0.36272
0.99532	0.48521	0.99435	0.47144	0.99280	0.43001	0.99186	0.42241
0.99386	0.55451	0.99258	0.53877	0.99057	0.49076	0.98932	0.48208
0.99220	0.62381	0.99056	0.60610	0.98804	0.55087	0.98643	0.54111
0.99032	0.69310	0.98826	0.67341	0.98518	0.61097	0.98315	0.60012
0.98822	0.76239	0.98569	0.74072	0.98201	0.67028	0.97949	0.65835
0.98590	0.83167	0.98283	0.80801	0.97848	0.72957	0.97538	0.71656
0.98334	0.90094	0.97966	0.87529	0.97462	0.78794	0.97086	0.77385
0.98053	0.97021	0.97615	0.94256	0.97034	0.84629	0.96579	0.83110
0.97745	1.03946	0.97229	1.00980	0.96569	0.90360	0.96021	0.88731
0.97409	1.10870	0.96804	1.07702	0.96051	0.96088	0.95388	0.94345
0.97043	1.17792	0.96335	1.14421	0.95486	1.01701	0.94679	0.99842
0.96643	1.24712	0.95817	1.21137	0.94842	1.07307	0.93841	1.05321
0.96207	1.31630	0.95242	1.27848	0.94112	1.12785	0.92850	1.10660
0.95726	1.38546	0.94597	1.34552	0.93252	1.18245	0.91595	1.15943
0.95196	1.45457	0.93869	1.41248	0.91875	1.23476	0.89328	1.20748
0.94599	1.52363	0.93019	1.47930	0.87976	1.27224	0.85716	1.24645
0.93919	1.59262	0.92012	1.54590	0.82820	1.28400	0.81209	1.27228
0.93108	1.66146	0.90716	1.61199	0.77595	1.29216	0.76139	1.28364
0.92049	1.72997	0.88862	1.67674	0.72465	1.29866	0.71140	1.29265
0.90641	1.79784	0.86020	1.73781	0.67320	1.30378	0.66101	1.29904
0.87209	1.85807	0.82260	1.79369	0.62281	1.30805	0.61161	1.30437
0.82087	1.90478	0.77673	1.84301	0.57237	1.31170	0.56213	1.30877
0.75775	1.93341	0.72373	1.88457	0.52294	1.31483	0.51362	1.31251
0.69030	1.94939	0.66488	1.91734	0.47349	1.31755	0.46507	1.31573
0.62206	1.96159	0.60165	1.94053	0.42496	1.31989	0.41742	1.31847
0.55336	1.97080	0.53614	1.95619	0.37641	1.32193	0.36975	1.32085
0.48446	1.97837	0.46987	1.96821	0.32865	1.32366	0.32284	1.32285
0.41542	1.98447	0.40316	1.97746	0.28088	1.32512	0.27591	1.32455
0.34629	1.98942	0.33622	1.98480	0.23371	1.32633	0.22959	1.32594
0.27709	1.99332	0.26911	1.99046	0.18655	1.32731	0.18326	1.32706
0.20785	1.99628	0.20191	1.99472	0.13981	1.32806	0.13734	1.32792
0.13858	1.99836	0.13464	1.99768	0.09306	1.32858	0.09142	1.32852
0.06929	1.99959	0.06733	1.99942	0.04653	1.32890	0.04571	1.32888
0.00000	2.00000	0.00000	2.00000	0.00000	1.32900	0.00000	1.32900

Table 7: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large plate with a hole close to an edge (e/w = 2) subject to uniaxial remote loading (S_2:S_1 = 1:0), optimised using the multi-peak method.*

Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1}	K_{t2}	K_{t3}	K_{t4}	μ_1	μ_2 (= μ_4)	μ_3	$\sum_{j=1}^{4}\mu_j$
Optimal shape											
1.000	1.085	0.02713	0.200	2.787	–0.996	2.432	–0.996	0.225	0.181	0.275	0.862
1.000	1.049	0.05670	0.400	2.940	–1.068	2.544	–1.068	0.188	0.150	0.238	0.725
2.000	2.117	—	0.100	2.239	–0.997	2.006	–0.997	0.338	0.119	0.375	0.950
2.000	2.078	—	0.200	2.262	–1.003	2.026	–1.003	0.325	0.106	0.363	0.900
2.000	2.068	0.02016	0.222	2.270	–1.011	2.030	–1.011	0.325	0.106	0.363	0.900
2.000	2.060	—	0.300	2.288	–1.022	2.039	–1.022	0.313	0.106	0.363	0.888
2.000	2.038	—	0.400	2.312	–1.048	2.062	–1.048	0.313	0.094	0.350	0.850
2.000	2.032	0.04018	0.430	2.319	–1.054	2.066	–1.054	0.300	0.088	0.350	0.825
2.000	2.024	—	0.600	2.339	–1.078	2.081	–1.078	0.300	0.081	0.338	0.800
3.000	3.118	—	0.100	2.046	–1.003	1.847	–1.003	0.388	0.088	0.413	0.961
3.000	3.102	—	0.130	2.047	–1.006	1.857	–1.006	0.384	0.088	0.413	0.961
3.000	3.039	0.02047	0.300	2.071	–1.038	1.871	–1.038	0.371	0.070	0.403	0.913
3.000	3.007	0.03990	0.570	2.105	–1.134	1.898	–1.134	0.347	0.046	0.382	0.821
3.000	3.006	—	0.600	2.109	–1.148	1.901	–1.148	0.347	0.042	0.382	0.814
Initial elliptical shape											
1.000[1]	—	0.07958	1.000	3.986	–1.314	3.391	–1.314	—	—	—	—
2.000	—	0.05175	0.502	3.089	–1.510	2.325	–1.510	—	—	—	—
3.000	—	0.02503	0.333	2.823	–1.672	1.981	–1.672	—	—	—	—

[1] Corresponds to a circular hole.

Table 8: *Coordinates of a ½-symmetric optimal hole close to the edge of a large plate (e/w = 2) under remote uniaxial loading (S_2:S_1 = 1:0), for holes of aspect ratio $h/w \approx 1$ and a variety of normalised minimum radius of curvatures (ρ_{min}/w), optimised using multi-peak method.*

h/w = 1.085 ρ_{min}/w = 0.200 ρ_{min}/u_T = 0.02713 Peak K_t = +2.787, +2.432 and –0.996				h/w = 1.049 ρ_{min}/w = 0.400 ρ_{min}/u_T = 0.05670 Peak K_t = +2.940, +2.544 and –1.068			
x/w	y/w	x/w	y/w	x/w	y/w	x/w	y/w
1.0000	0.0000	–0.0179	1.0034	1.0000	0.0000	–0.0206	1.0029
0.9997	0.0461	–0.0638	0.9989	0.9997	0.0441	–0.0645	0.9987
0.9988	0.0921	–0.1097	0.9946	0.9987	0.0882	–0.1084	0.9946
0.9973	0.1382	–0.1555	0.9906	0.9970	0.1322	–0.1523	0.9906
0.9952	0.1842	–0.2014	0.9868	0.9947	0.1763	–0.1962	0.9867
0.9924	0.2302	–0.2474	0.9832	0.9917	0.2202	–0.2401	0.9828
0.9891	0.2761	–0.2933	0.9799	0.9880	0.2642	–0.2841	0.9791
0.9851	0.3220	–0.3393	0.9768	0.9836	0.3080	–0.3280	0.9754
0.9804	0.3678	–0.3853	0.9740	0.9785	0.3518	–0.3719	0.9717
0.9751	0.4136	–0.4313	0.9715	0.9726	0.3955	–0.4159	0.9680
0.9690	0.4593	–0.4773	0.9694	0.9659	0.4391	–0.4598	0.9644
0.9623	0.5048	–0.5233	0.9675	0.9583	0.4826	–0.5037	0.9605
0.9547	0.5503	–0.5694	0.9661	0.9499	0.5258	–0.5476	0.9564
0.9464	0.5956	–0.6154	0.9648	0.9406	0.5689	–0.5914	0.9514
0.9372	0.6407	–0.6615	0.9641	0.9302	0.6118	–0.6350	0.9447
0.9271	0.6857	–0.7075	0.9626	0.9188	0.6544	–0.6776	0.9333
0.9160	0.7304	–0.7536	0.9624	0.9060	0.6966	–0.7187	0.9173
0.9037	0.7748	–0.7984	0.9516	0.8918	0.7383	–0.7578	0.8969
0.8902	0.8188	–0.8395	0.9309	0.8758	0.7794	–0.7944	0.8723
0.8752	0.8624	–0.8748	0.9013	0.8577	0.8196	–0.8280	0.8438
0.8584	0.9053	–0.9025	0.8645	0.8366	0.8583	–0.8584	0.8118
0.8393	0.9472	–0.9209	0.8223	0.8113	0.8944	–0.8850	0.7767
0.8171	0.9876	–0.9355	0.7786	0.7823	0.9276	–0.9076	0.7388
0.7893	1.0243	–0.9468	0.7339	0.7498	0.9574	–0.9259	0.6987
0.7539	1.0538	–0.9562	0.6888	0.7142	0.9835	–0.9397	0.6568
0.7127	1.0743	–0.9638	0.6434	0.6760	1.0055	–0.9508	0.6142
0.6678	1.0849	–0.9704	0.5978	0.6356	1.0232	–0.9599	0.5710
0.6218	1.0850	–0.9759	0.5520	0.5936	1.0364	–0.9674	0.5276
0.5761	1.0789	–0.9806	0.5062	0.5503	1.0449	–0.9738	0.4840
0.5304	1.0728	–0.9845	0.4603	0.5064	1.0486	–0.9791	0.4402
0.4849	1.0661	–0.9879	0.4144	0.4623	1.0475	–0.9837	0.3963
0.4393	1.0595	–0.9907	0.3684	0.4183	1.0449	–0.9875	0.3524
0.3937	1.0530	–0.9931	0.3224	0.3743	1.0413	–0.9907	0.3084
0.3480	1.0467	–0.9950	0.2763	0.3304	1.0373	–0.9933	0.2644
0.3024	1.0406	–0.9966	0.2303	0.2865	1.0331	–0.9954	0.2204
0.2567	1.0347	–0.9979	0.1843	0.2427	1.0288	–0.9971	0.1763
0.2110	1.0289	–0.9988	0.1382	0.1988	1.0244	–0.9984	0.1323
0.1652	1.0234	–0.9995	0.0921	0.1549	1.0200	–0.9993	0.0882
0.1195	1.0181	–0.9999	0.0461	0.1110	1.0157	–0.9998	0.0441
0.0737	1.0130	–1.0000	0.0000	0.0672	1.0113	–1.0000	0.0000
0.0279	1.0081			0.0233	1.0071		

Table 9: *Coordinates of a ½-symmetric optimal hole close to the edge of a large plate (e/w = 2) under remote uniaxial loading (S_2:S_1 = 1:0), for holes of aspect ratio $h/w \approx 2$ and a variety of normalised minimum radius of curvatures (ρ_{min}/w), optimised using multi-peak method.*

h/w = 2.068 ρ_{min}/w = 0.222 ρ_{min}/u_T = 0.02016 Peak K_t = +2.270, +2.030 and –1.011				h/w = 2.032 ρ_{min}/w = 0.430 ρ_{min}/u_T = 0.04018 Peak K_t = +2.319, +2.066 and –1.054			
x/w	y/w	x/w	y/w	x/w	y/w	x/w	y/w
1.0000	0.0000	–0.0922	1.9946	1.0000	0.0000	–0.0975	1.9927
0.9997	0.0688	–0.1606	1.9867	0.9997	0.0669	–0.1639	1.9851
0.9989	0.1377	–0.2289	1.9791	0.9988	0.1338	–0.2303	1.9774
0.9975	0.2065	–0.2974	1.9719	0.9974	0.2006	–0.2968	1.9696
0.9955	0.2753	–0.3659	1.9651	0.9954	0.2675	–0.3632	1.9618
0.9930	0.3441	–0.4344	1.9587	0.9928	0.3343	–0.4296	1.9537
0.9899	0.4128	–0.5030	1.9528	0.9895	0.4011	–0.4959	1.9452
0.9862	0.4815	–0.5716	1.9474	0.9857	0.4679	–0.5621	1.9357
0.9819	0.5502	–0.6402	1.9422	0.9813	0.5347	–0.6279	1.9234
0.9770	0.6189	–0.7088	1.9365	0.9762	0.6014	–0.6909	1.9012
0.9715	0.6875	–0.7767	1.9251	0.9705	0.6680	–0.7498	1.8694
0.9653	0.7561	–0.8378	1.8935	0.9642	0.7346	–0.8030	1.8289
0.9585	0.8246	–0.8863	1.8446	0.9571	0.8011	–0.8493	1.7806
0.9510	0.8930	–0.9175	1.7833	0.9493	0.8675	–0.8875	1.7257
0.9428	0.9613	–0.9369	1.7172	0.9409	0.9339	–0.9168	1.6656
0.9339	1.0296	–0.9508	1.6498	0.9316	1.0001	–0.9364	1.6016
0.9243	1.0977	–0.9613	1.5818	0.9216	1.0663	–0.9504	1.5362
0.9138	1.1658	–0.9695	1.5135	0.9107	1.1322	–0.9609	1.4702
0.9025	1.2336	–0.9759	1.4449	0.8988	1.1981	–0.9692	1.4038
0.8903	1.3014	–0.9810	1.3763	0.8861	1.2637	–0.9756	1.3372
0.8771	1.3689	–0.9851	1.3076	0.8723	1.3292	–0.9807	1.2705
0.8629	1.4363	–0.9883	1.2388	0.8573	1.3944	–0.9848	1.2038
0.8475	1.5034	–0.9909	1.1700	0.8411	1.4593	–0.9881	1.1369
0.8309	1.5702	–0.9930	1.1012	0.8235	1.5238	–0.9907	1.0701
0.8129	1.6366	–0.9946	1.0324	0.8042	1.5879	–0.9927	1.0033
0.7933	1.7026	–0.9958	0.9636	0.7831	1.6513	–0.9944	0.9364
0.7716	1.7679	–0.9968	0.8948	0.7595	1.7139	–0.9957	0.8695
0.7477	1.8325	–0.9976	0.8260	0.7328	1.7752	–0.9967	0.8026
0.7204	1.8957	–0.9982	0.7571	0.7014	1.8343	–0.9975	0.7358
0.6882	1.9565	–0.9987	0.6883	0.6612	1.8877	–0.9981	0.6689
0.6456	2.0106	–0.9990	0.6195	0.6132	1.9343	–0.9986	0.6020
0.5885	2.0490	–0.9993	0.5506	0.5586	1.9729	–0.9990	0.5351
0.5224	2.0681	–0.9995	0.4818	0.4986	2.0026	–0.9993	0.4682
0.4536	2.0661	–0.9997	0.4130	0.4348	2.0226	–0.9995	0.4013
0.3853	2.0580	–0.9998	0.3442	0.3686	2.0324	–0.9997	0.3344
0.3171	2.0486	–0.9999	0.2753	0.3018	2.0319	–0.9998	0.2675
0.2489	2.0391	–0.9999	0.2065	0.2350	2.0274	–0.9999	0.2007
0.1808	2.0296	–1.0000	0.1377	0.1684	2.0214	–1.0000	0.1338
0.1126	2.0204	–1.0000	0.0688	0.1019	2.0147	–1.0000	0.0669
0.0444	2.0115	–1.0000	0.0000	0.0354	2.0076	–1.0000	0.0000
-0.0239	2.0029			-0.0310	2.0002		

Table 10: *Coordinates of a ½-symmetric optimal hole close to the edge of a large plate (e/w = 2) under remote uniaxial loading (S_2:S_1 = 1:0), for holes of aspect ratio $h/w \approx 3$ and a variety of normalised minimum radius of curvatures (ρ_{min}/w), optimised using multi-peak method.*

h/w = 3.039 ρ_{min}/w = 0.300 ρ_{min}/u_T = 0.02047 Peak K_t = +2.071, +1.871 and –1.038				h/w = 3.007 ρ_{min}/w = 0.570 ρ_{min}/u_T = 0.03990 Peak K_t = +2.105, +1.898 and –1.134			
x/w	y/w	x/w	y/w	x/w	y/w	x/w	y/w
1.0000	0.0000	–0.1219	2.9902	1.0000	0.0000	–0.1334	2.9905
0.9994	0.1291	–0.1650	2.9850	0.9994	0.1258	–0.1755	2.9859
0.9977	0.2582	–0.2101	2.9795	0.9977	0.2516	–0.2194	2.9806
0.9949	0.3865	–0.2552	2.9741	0.9947	0.3767	–0.2634	2.9749
0.9909	0.5148	–0.3041	2.9683	0.9906	0.5018	–0.3109	2.9683
0.9858	0.6417	–0.3530	2.9626	0.9853	0.6254	–0.3583	2.9611
0.9795	0.7685	–0.4069	2.9563	0.9788	0.7490	–0.4106	2.9525
0.9722	0.8932	–0.4609	2.9501	0.9713	0.8705	–0.4626	2.9428
0.9636	1.0178	–0.5208	2.9433	0.9624	0.9919	–0.5200	2.9302
0.9541	1.1395	–0.5807	2.9364	0.9526	1.1105	–0.5763	2.9134
0.9434	1.2611	–0.6469	2.9280	0.9414	1.2290	–0.6363	2.8882
0.9317	1.3792	–0.7128	2.9173	0.9293	1.3440	–0.6930	2.8563
0.9187	1.4972	–0.7812	2.8904	0.9158	1.4589	–0.7508	2.8140
0.9049	1.6110	–0.8409	2.8476	0.9014	1.5698	–0.8028	2.7648
0.8897	1.7247	–0.8922	2.7859	0.8857	1.6804	–0.8521	2.7041
0.8739	1.8336	–0.9253	2.7129	0.8691	1.7864	–0.8926	2.6373
0.8565	1.9422	–0.9465	2.6287	0.8509	1.8921	–0.9257	2.5594
0.8386	2.0456	–0.9608	2.5430	0.8321	1.9926	–0.9469	2.4774
0.8191	2.1487	–0.9717	2.4504	0.8116	2.0927	–0.9620	2.3878
0.7991	2.2459	–0.9795	2.3575	0.7904	2.1871	–0.9725	2.2976
0.7774	2.3427	–0.9856	2.2585	0.7673	2.2811	–0.9805	2.2011
0.7552	2.4333	–0.9899	2.1593	0.7435	2.3688	–0.9862	2.1046
0.7310	2.5233	–0.9931	2.0545	0.7173	2.4558	–0.9905	2.0024
0.7063	2.6066	–0.9954	1.9496	0.6904	2.5360	–0.9935	1.9002
0.6791	2.6890	–0.9970	1.8396	0.6600	2.6150	–0.9957	1.7930
0.6511	2.7642	–0.9981	1.7295	0.6279	2.6863	–0.9972	1.6858
0.6190	2.8377	–0.9988	1.6149	0.5889	2.7541	–0.9982	1.5740
0.5841	2.9024	–0.9993	1.5002	0.5454	2.8109	–0.9989	1.4623
0.5371	2.9588	–0.9996	1.3815	0.4951	2.8619	–0.9993	1.3466
0.4837	2.9988	–0.9998	1.2628	0.4442	2.9025	–0.9996	1.2309
0.4227	3.0260	–0.9999	1.1407	0.3891	2.9369	–0.9998	1.1119
0.3637	3.0384	–0.9999	1.0186	0.3361	2.9625	–0.9999	0.9929
0.3035	3.0388	–1.0000	0.8938	0.2809	2.9824	–0.9999	0.8711
0.2493	3.0346	–1.0000	0.7689	0.2296	2.9954	–1.0000	0.7494
0.1953	3.0289	–1.0000	0.6419	0.1772	3.0036	–1.0000	0.6257
0.1464	3.0233	–1.0000	0.5149	0.1294	3.0069	–1.0000	0.5019
0.0975	3.0174	–1.0000	0.3866	0.0814	3.0066	–1.0000	0.3768
0.0524	3.0118	–1.0000	0.2582	0.0371	3.0048	–1.0000	0.2516
0.0073	3.0063	–1.0000	0.1291	–0.0071	3.0020	–1.0000	0.1258
–0.0358	3.0009	–1.0000	0.0000	–0.0492	2.9987	–1.0000	0.0000
–0.0788	2.9956			–0.0913	2.9948		

Table 11: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large plate with a central hole and inclined boundary constraint line, subject to uniaxial loading, optimised using the multi-peak method.*

b/r	Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1}	K_{t2}	K_{t3}	K_{t4}	μ_1	μ_2	μ_3	μ_4	$\sum_{j=1}^{4}\mu_j$
Optimal shape													
0.0	1.000	1.043	0.01986	0.147	2.329	–0.813	2.135	–0.784	0.156	0.181	0.269	0.169	0.775
0.0	1.000	1.014	0.04023	0.288	2.389	–0.811	2.209	–0.788	0.150	0.169	0.244	0.156	0.719
0.0	1.000	0.997	–	0.480	2.484	–0.823	2.315	–0.808	0.138	0.131	0.213	0.119	0.601
2.0	1.537	1.526	0.02032	0.188	2.558	–0.820	1.847	–0.727	0.093	0.153	0.322	0.093	0.661
2.0	1.537	1.487	0.03985	0.356	2.665	–0.825	1.899	–0.758	0.085	0.119	0.305	0.076	0.585
4.0	1.884	1.789	0.01978	0.206	2.814	–0.842	1.746	–0.731	0.067	0.134	0.353	0.061	0.616
4.0	1.884	1.758	0.03995	0.408	2.998	–0.848	1.786	–0.780	0.043	0.110	0.334	0.043	0.529
Initial shape													
0.0	1.000	–	0.097	–	3.096	–1.011	3.098	–1.013	–	–	–	–	–
2.0	1.537	–	0.159	–	3.024	–1.019	3.023	–1.019	–	–	–	–	–
4.0	1.884	–	0.070	–	3.288	–1.045	3.294	–1.051	–	–	–	–	–

Table 12: *Optimal K_t values and proportions of the hole boundary with flat-topped stress distribution for a large thick 3D plate with holes of various aspect ratios subject to uniaxial loading, optimised using the multi-peak method.*

Initial h/w	Optimal h/w	ρ_{min}/u_T	ρ_{min}/w	K_{t1} $(=K_{t3})$	K_{t2} $(=K_{t4})$	μ_1 $(=\mu_3)$	μ_2 $(=\mu_4)$	$\sum_{j=1}^{4}\mu_j$
Optimal shape								
1.000	1.036	0.01999	0.147	2.252	–0.899	0.263	0.188	0.900
1.000	1.019	0.04001	0.287	2.315	–0.905	0.250	0.163	0.825
1.000	1.000	0.08410	0.572	2.503	–0.955	0.188	0.125	0.625
2.000	2.022	0.02007	0.217	1.648	–0.802	0.363	0.100	0.925
2.000	2.001	0.03993	0.420	1.678	–0.843	0.338	0.075	0.825
Initial shape								
1.000	–	0.159	0.500	3.074	–1.118	–	–	–
2.000	–	0.052	0.250	1.941	–1.153	–	–	–

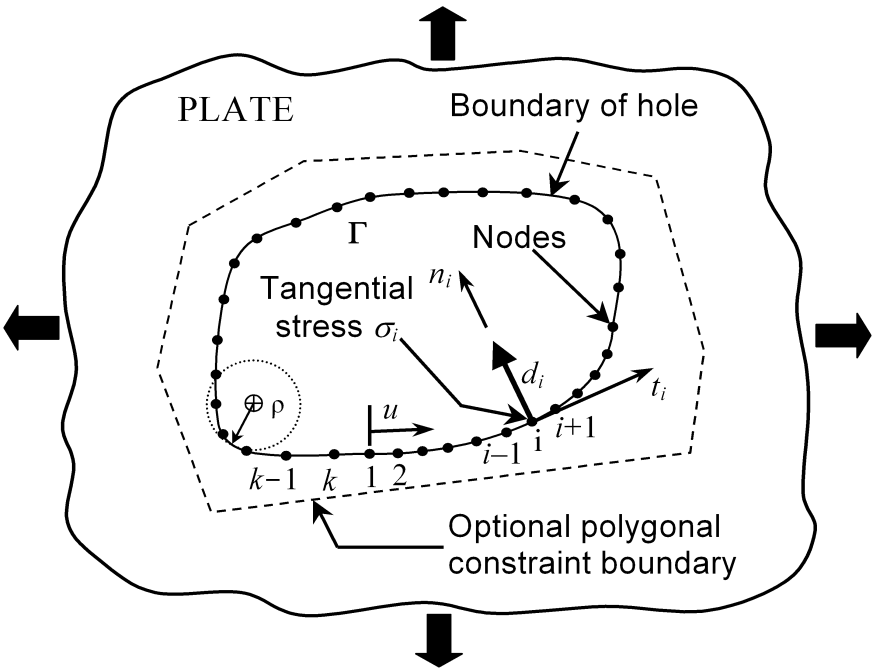


Figure 1: *Notation for geometrically constrained shape optimisation of a hole in a remotely loaded plate.*

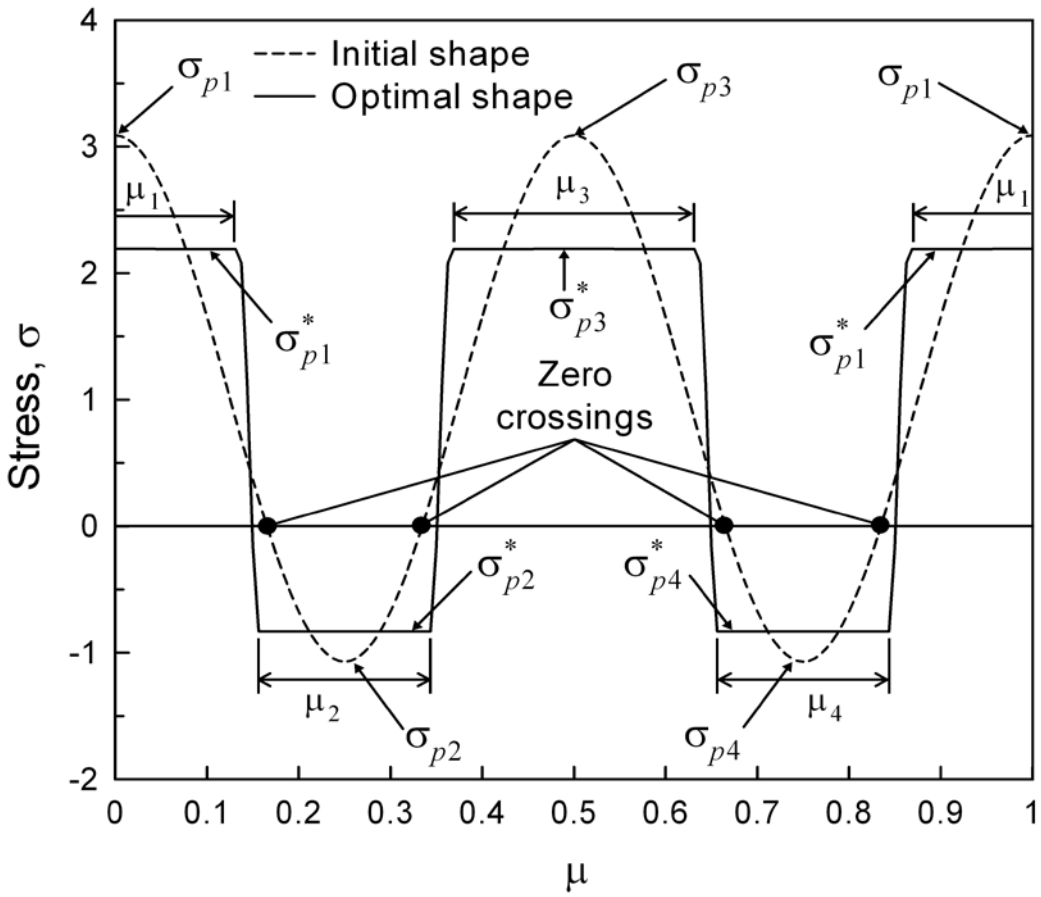
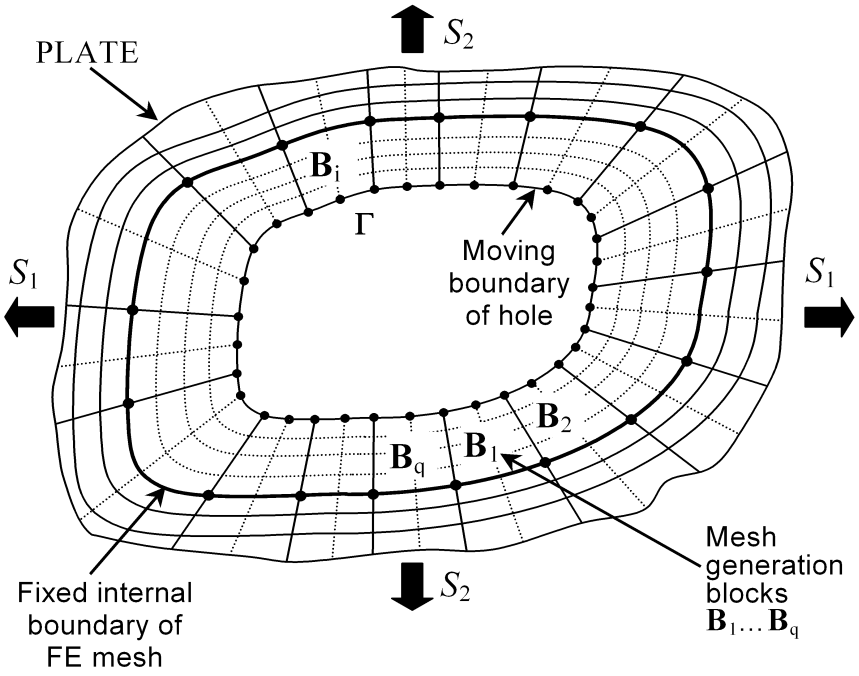
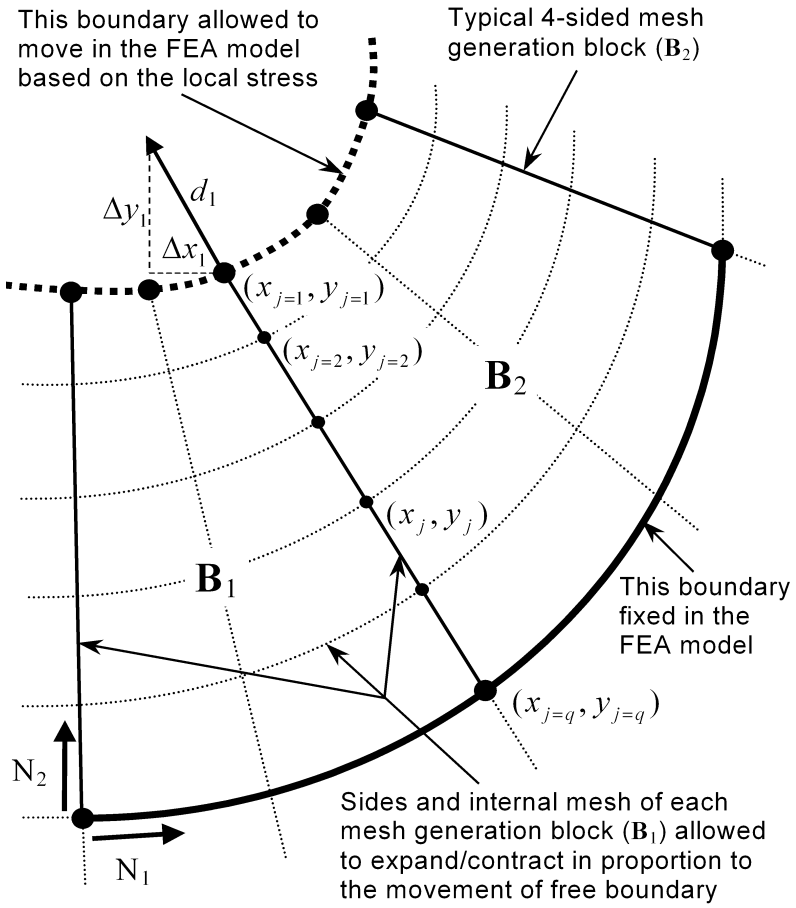


Figure 2: *Typical idealised multi-peak stress distribution around a circular hole in a uniaxially-loaded plate for an initial shape and the resulting optimal shape.*



(a)



(b)

Figure 3: *Use of automatic mesh generation blocks $\mathbf{B}_1 \ldots \mathbf{B}_k$ to define FEA mesh for shape optimisation of a hole in a plate. (a) General layout. (b) Detail view with notation.*

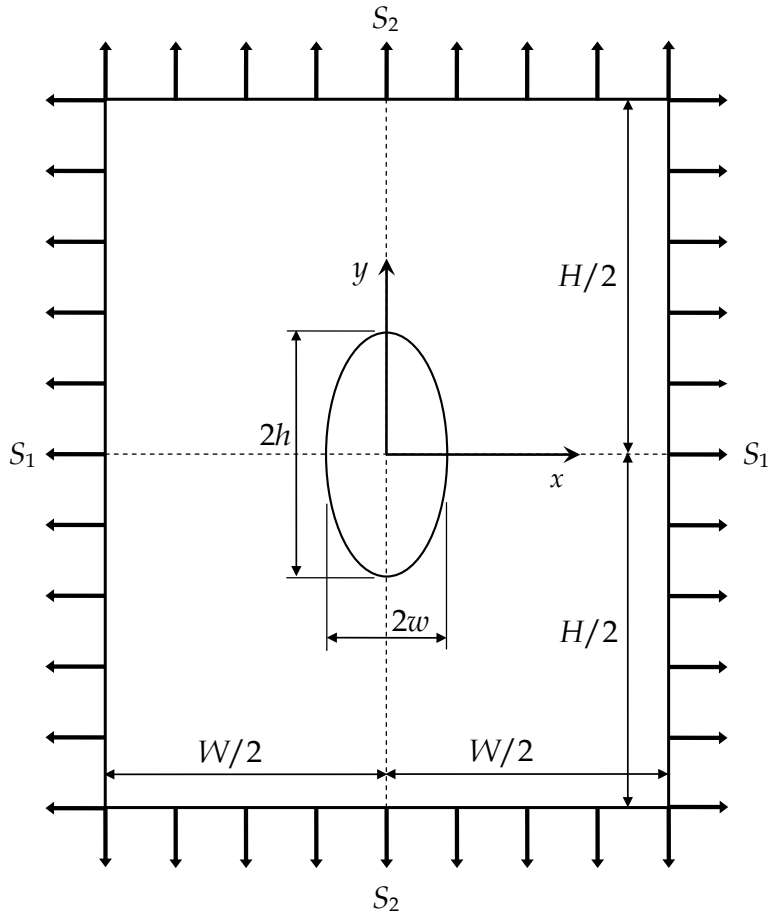


Figure 4: *Idealised geometry and loading for a centrally-located hole with aspect ratio h:w in a biaxially-loaded rectangular plate.*

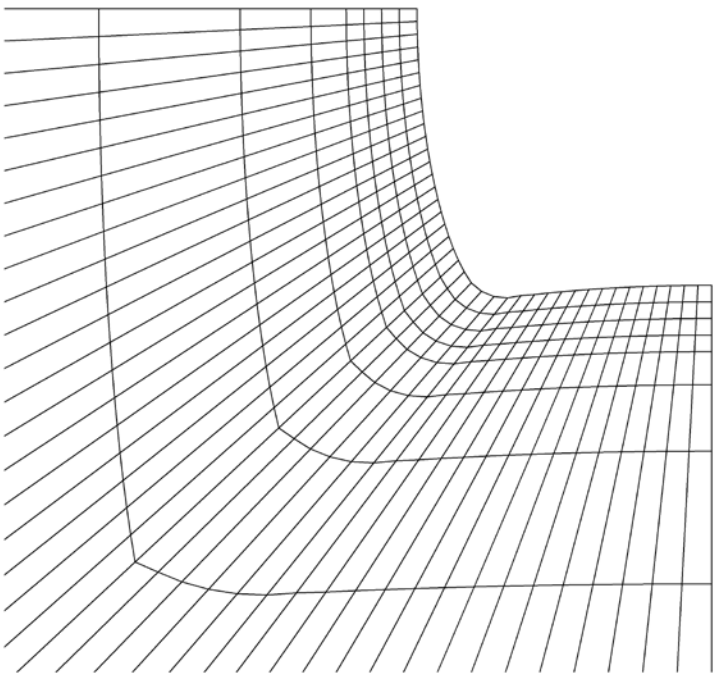
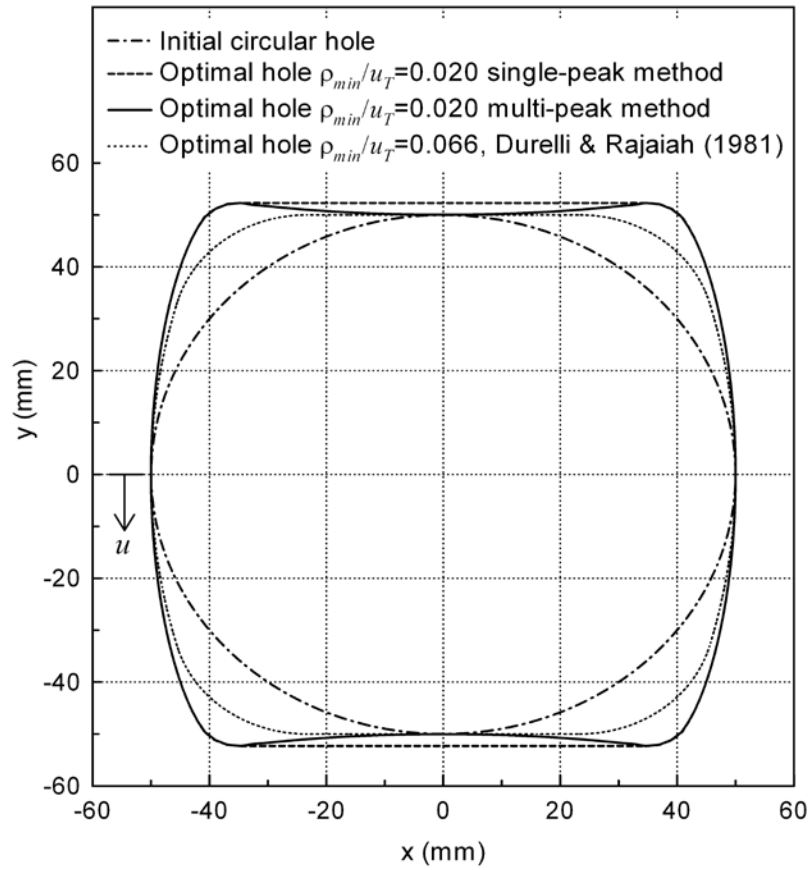
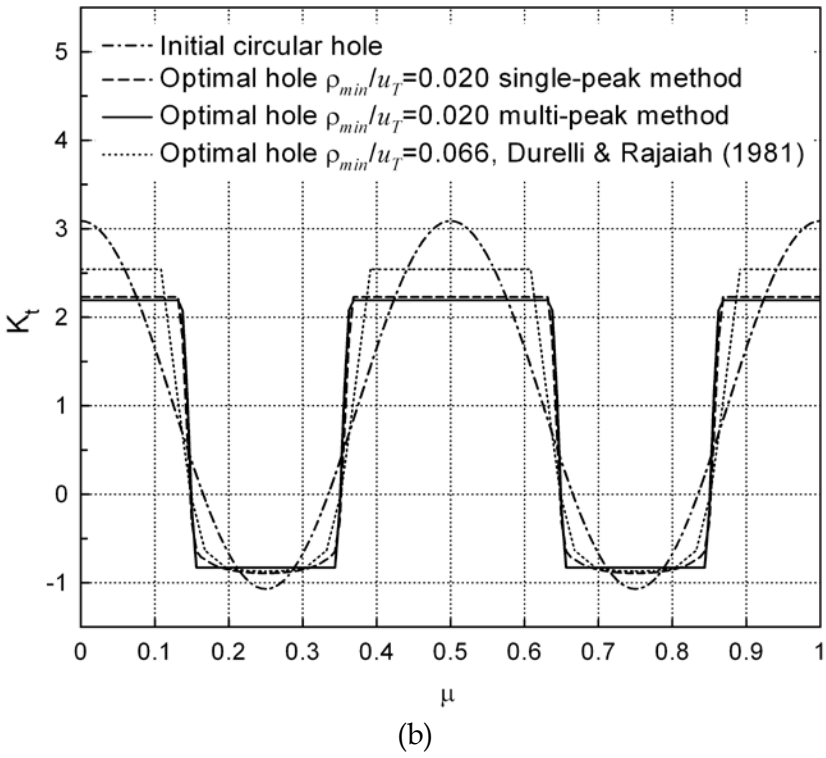


Figure 5: *Local detail of ¼-symmetry FEA mesh for typical optimal shape for a uniaxially-loaded (S_2:S_1 = 1:0) centrally-located hole with aspect ratio h:w = 1:1 and ρ_{min}/u_T = 0.02.*

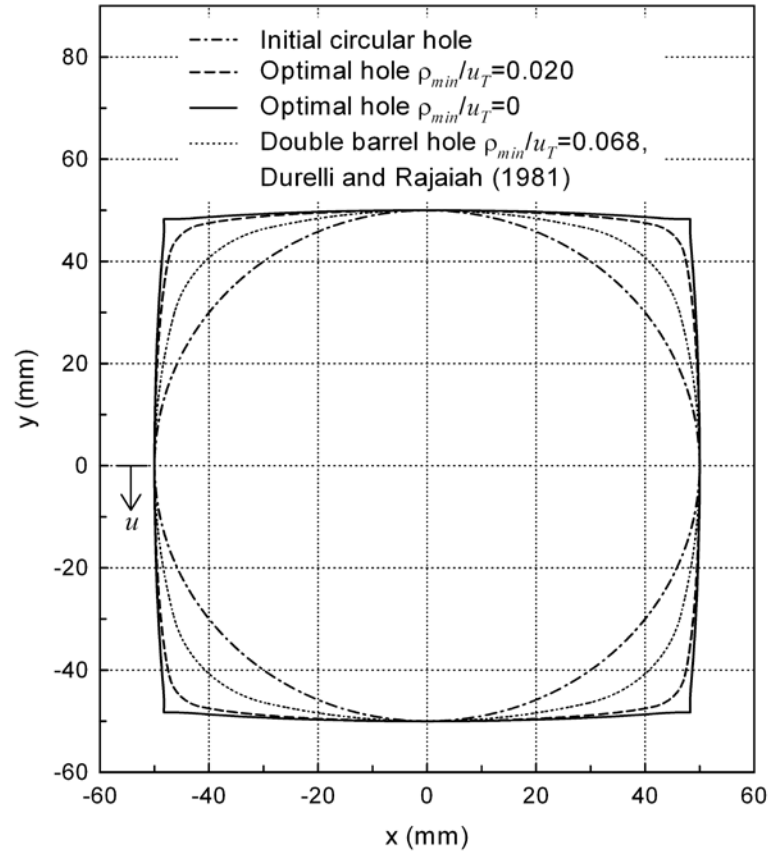


(a)

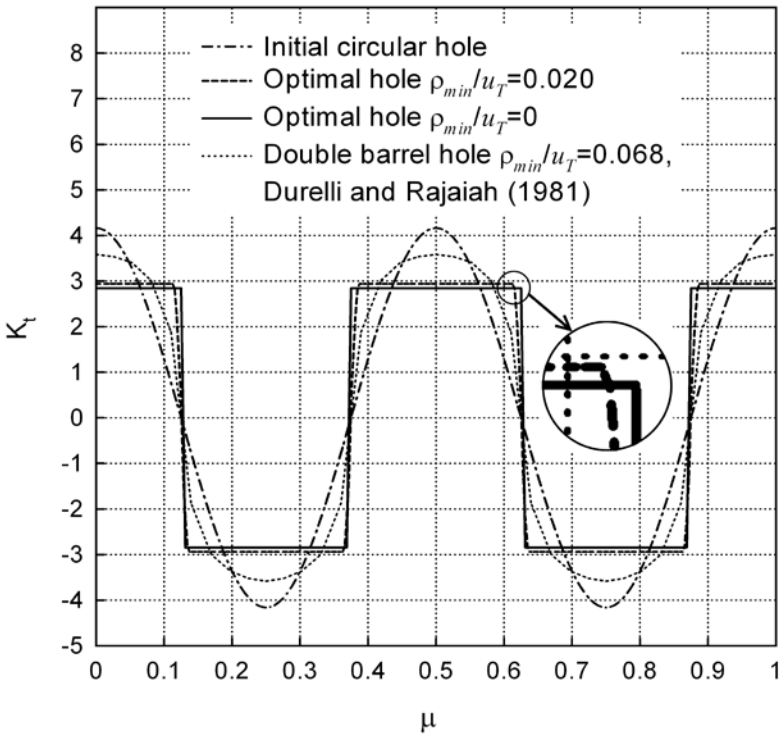


(b)

Figure 6: *Shape optimisation results for a h:w ≈ 1:1 aspect ratio central hole in a uniaxially-loaded (S_2:S_1 = 1:0) large square plate. (a) Initial and optimal shapes. (b) K_t around the hole perimeter for initial and optimal shapes.*



(a)



(b)

Figure 7: *Multi-peak shape optimisation results for a h:w = 1:1 aspect ratio central hole in a reverse biaxially-loaded (S_2:S_1 = 1:–1) large square plate. (a) Initial and optimal shapes. (b) K_t around hole perimeter for initial and optimal shapes.*

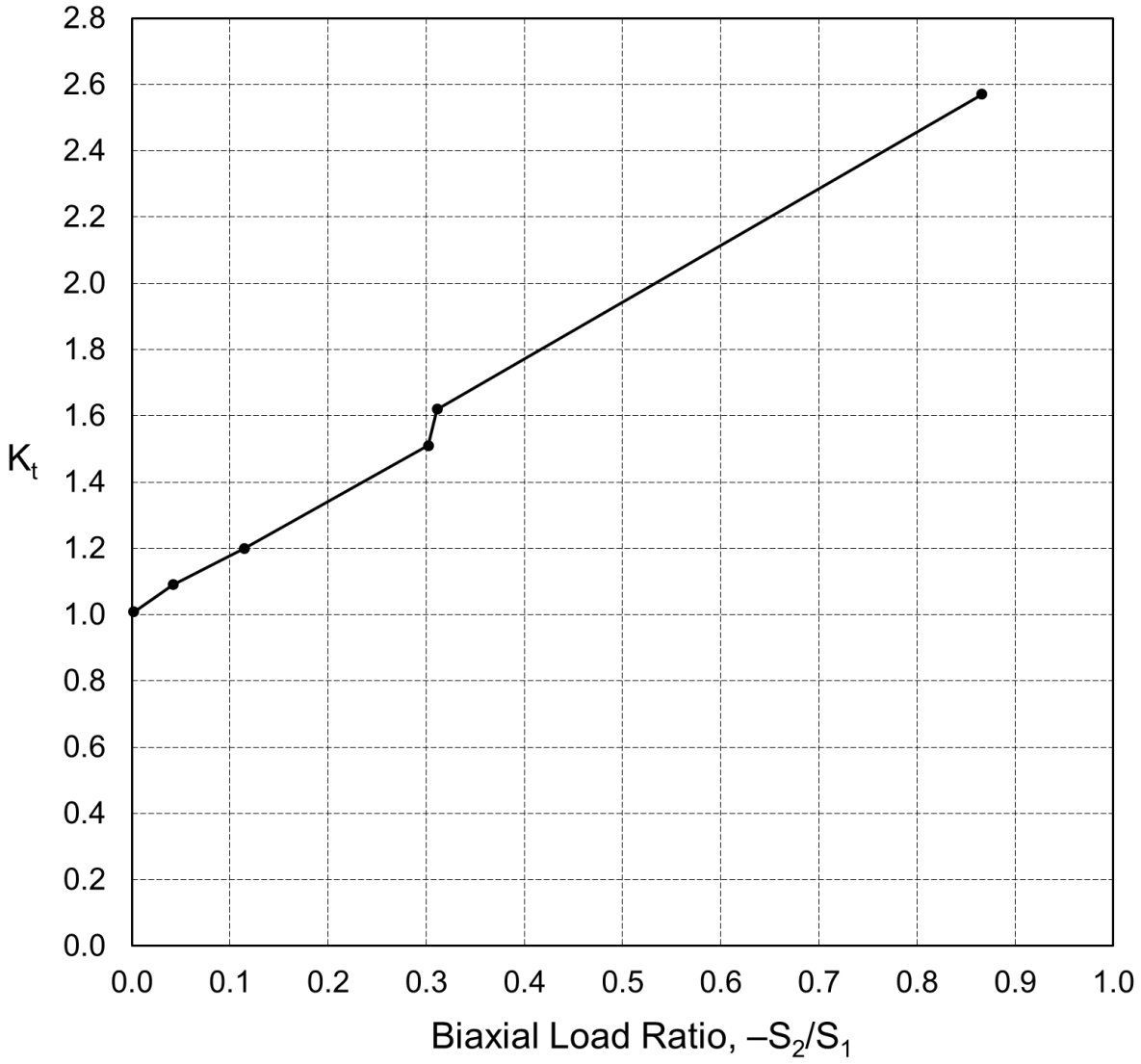
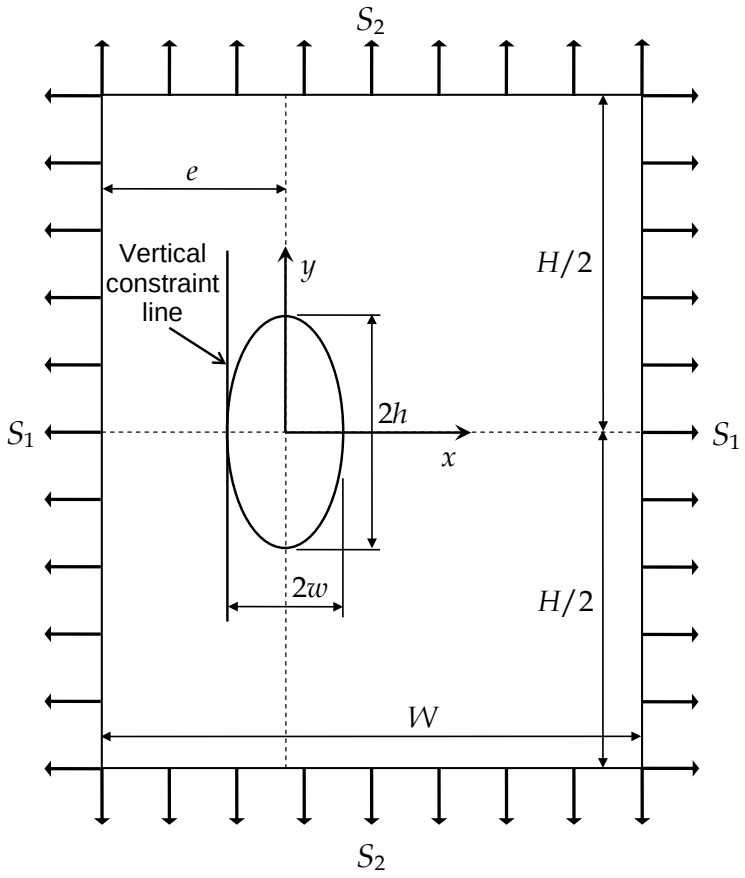
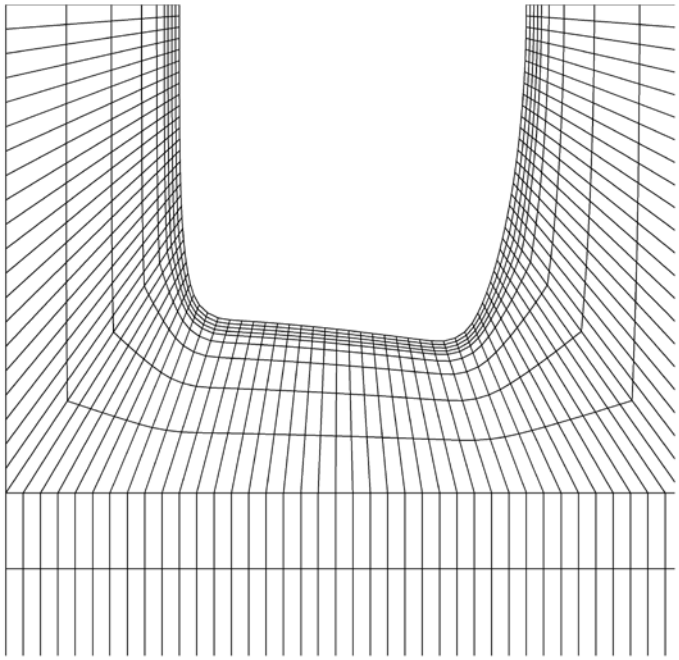


Figure 8: *K_t values for a single optimal hole in an infinite plate for different ratios of biaxial load ratio $-S_2/S_1$. Note that each value of K_t corresponds to a particular hole aspect ratio, as provided in Table 1 of Vigdergauz and Cherkayev (1986).*

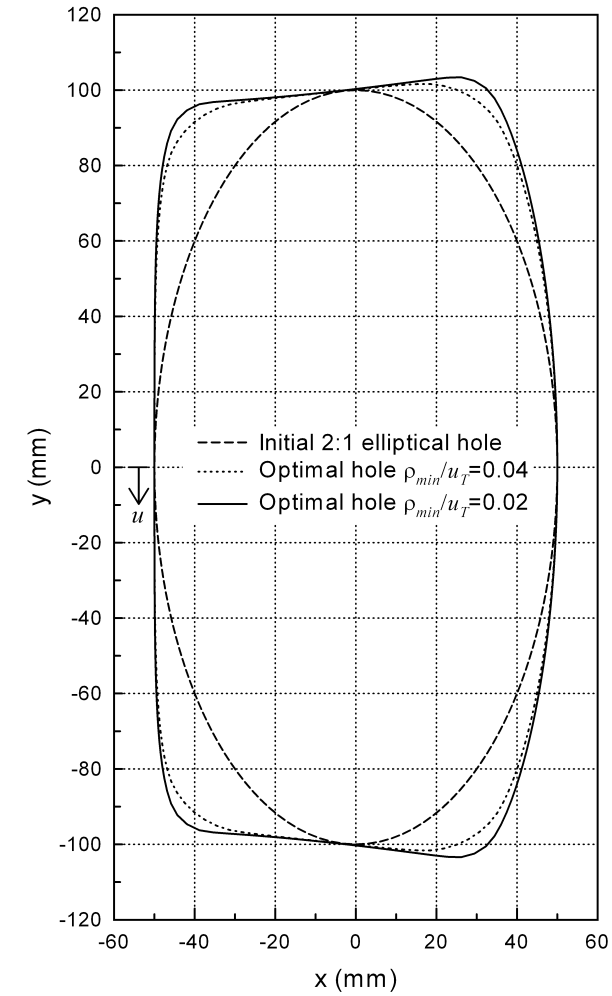


(a)

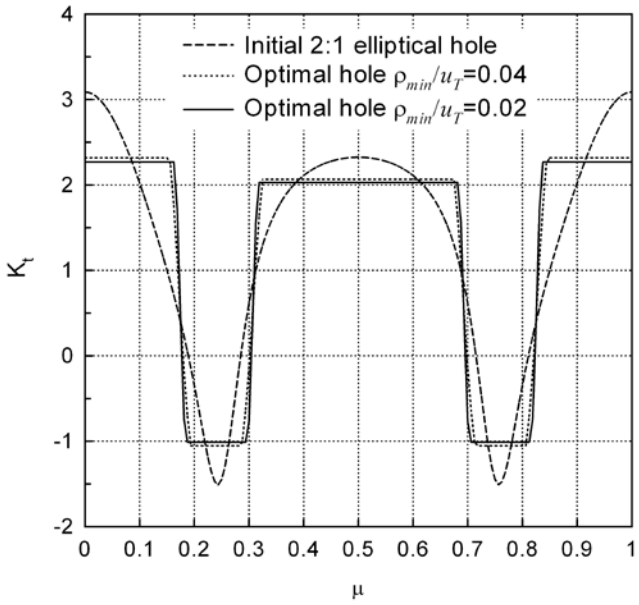


(b)

Figure 9: *Hole close to an edge of a uniaxially-loaded rectangular plate. (a) Generic hole of aspect ratio h:w. (b) Local detail of ½-symmetry FEA mesh for typical optimal shape (h:w ≈ 2:1 and $\rho_{min}/u_T = 0.020$).*



(a)



(b)

Figure 10: *Multi-peak shape optimisation results for a h:w ≈ 2:1 aspect ratio hole close to an edge (e/w = 2) in a uniaxially-loaded large rectangular plate. (a) Initial and optimal shapes. (b) K_t around the perimeter of the hole for initial and optimal shapes.*

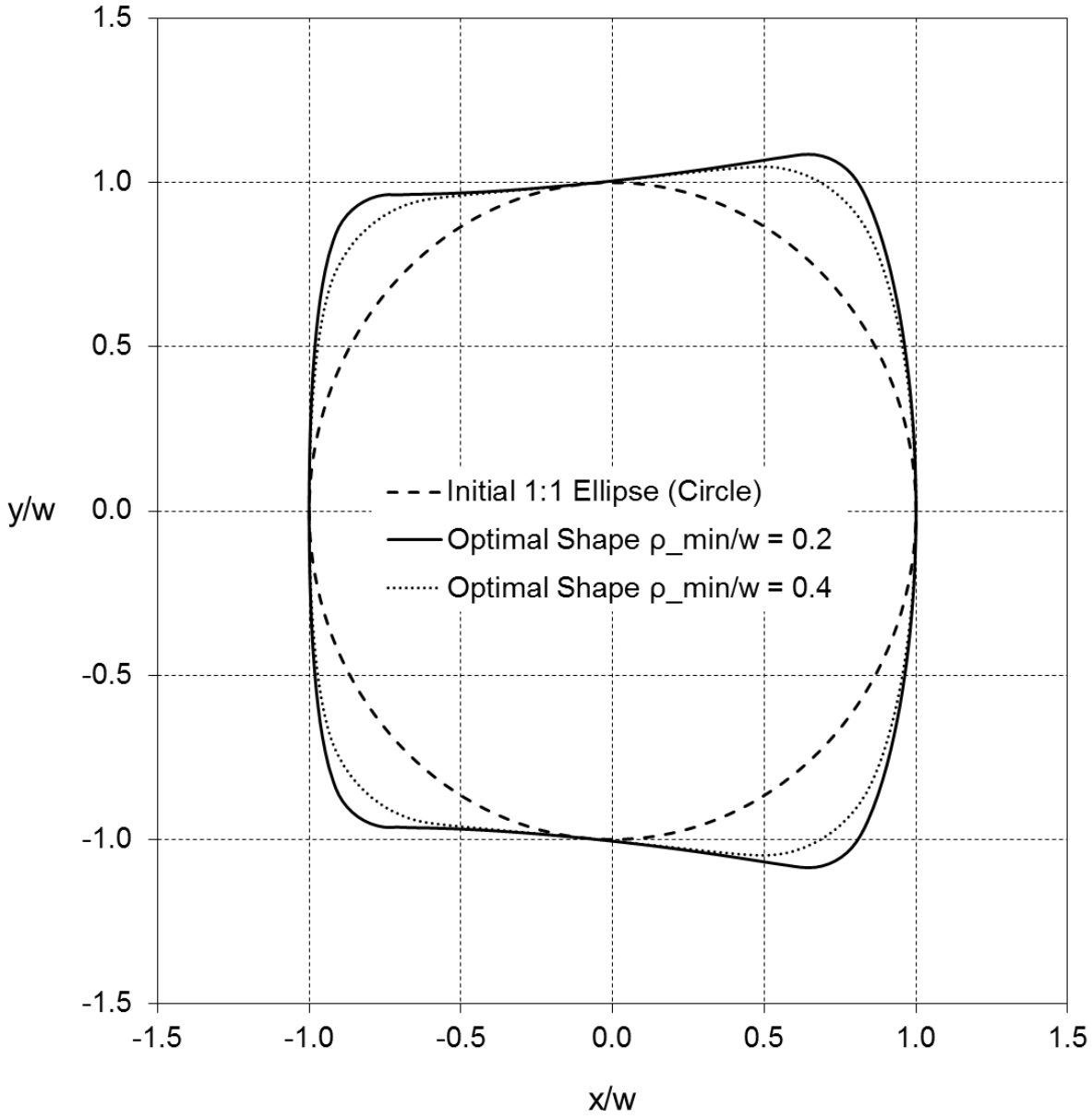


Figure 11: *Optimal shape obtained using multi-peak shape optimisation for a hole of aspect ratio h:w ≈1:1 close to an edge (e/w = 2) in a uniaxially-loaded large rectangular plate.*

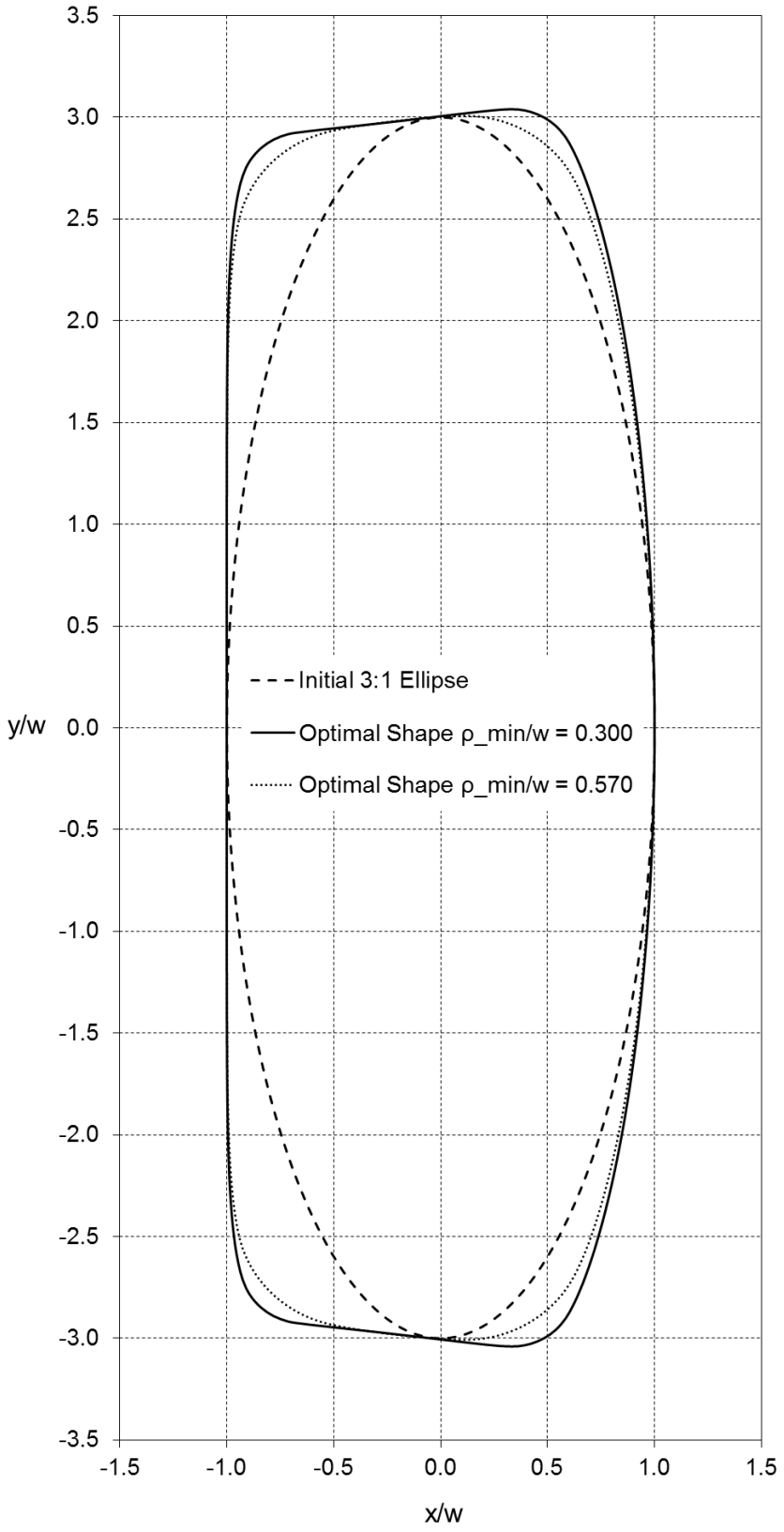
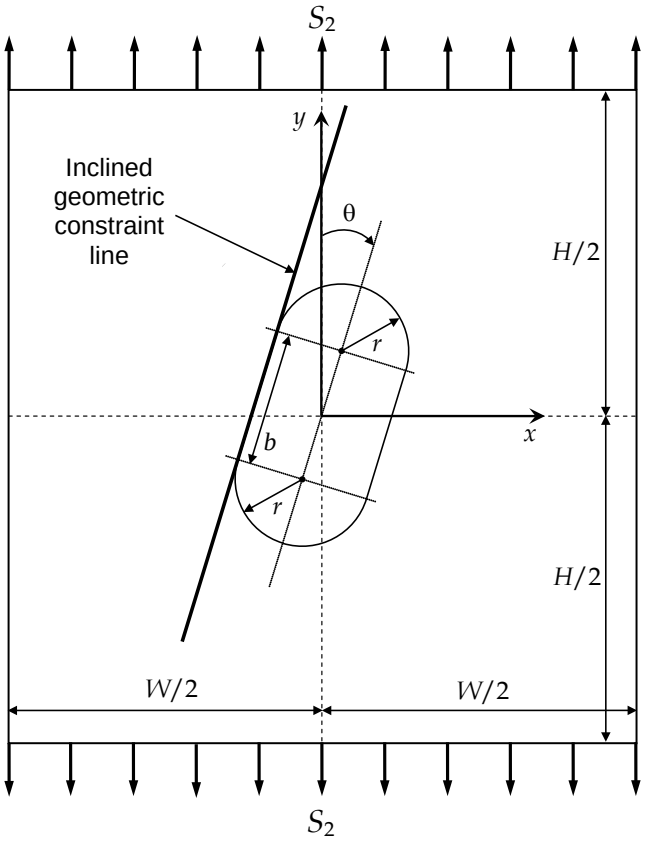
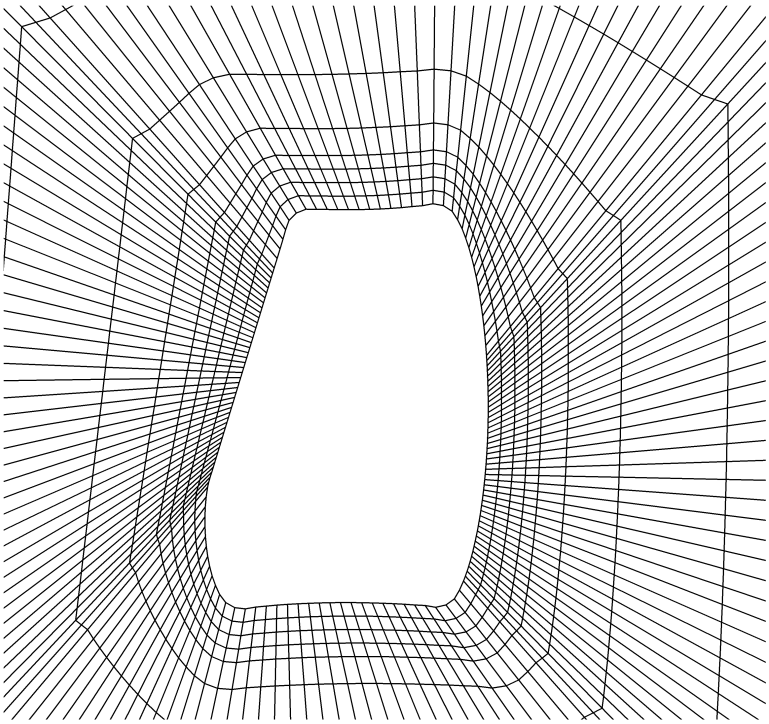


Figure 12: *Optimal shape obtained using multi-peak shape optimisation for a hole of aspect ratio h:w ≈3:1 close to an edge (e/w = 2) in a uniaxially-loaded large rectangular plate.*

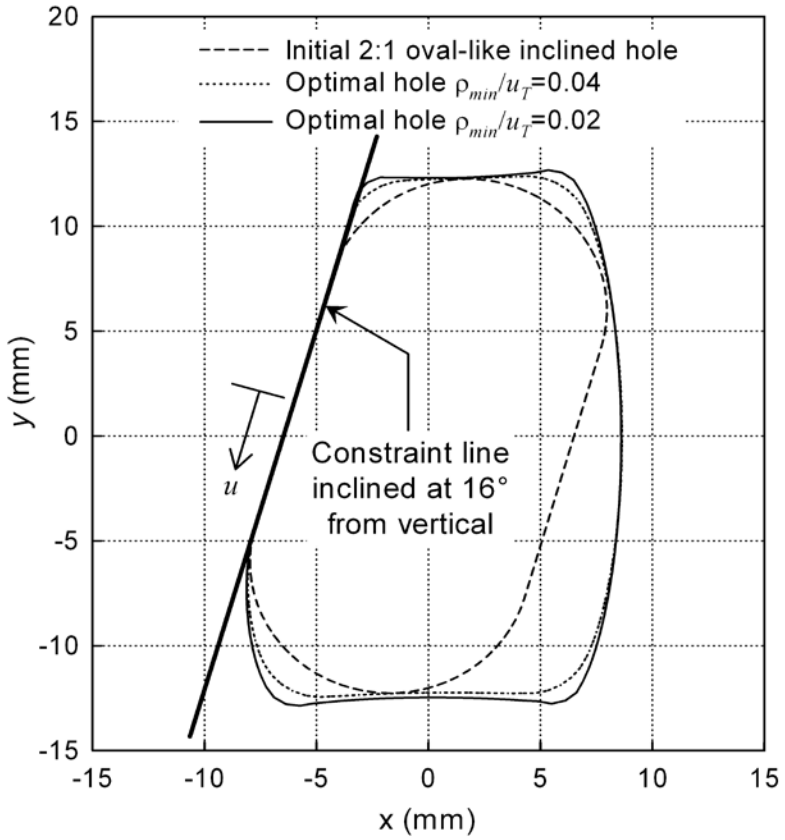


(a)

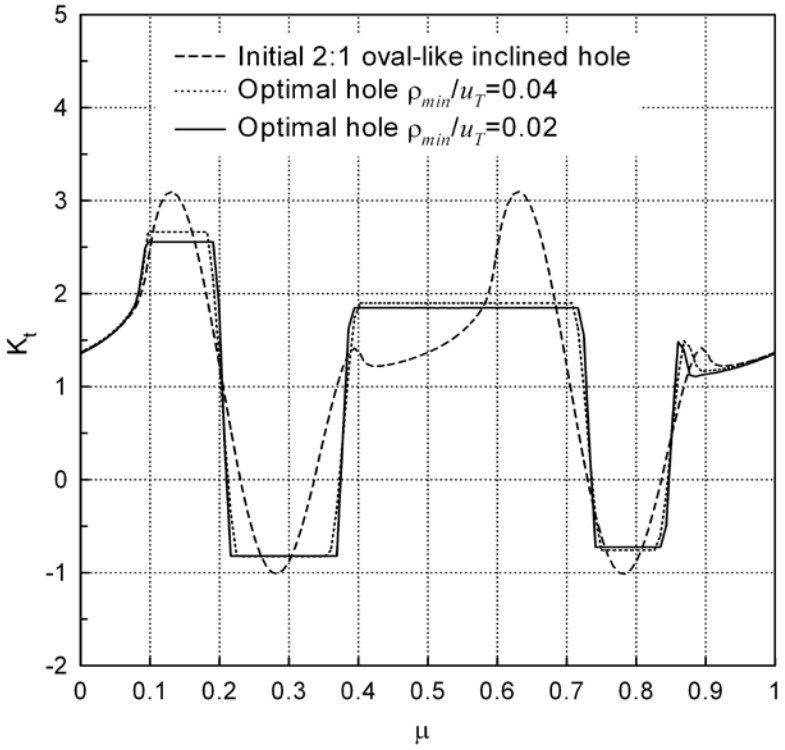


(b)

Figure 13: *Generic inclined slotted hole in a rectangular plate under uniaxial loading subjected to an inclined geometric constraint during shape optimisation (based on F-111 work, Heller et al. 1999 and Heller et al. 2002). (a) Inclined slotted hole. (b) Local detail of FEA mesh for typical optimal shape ($b/r = 2$ and $\rho_{min}/u_T = 0.02$).*



(a)



(b)

Figure 14: *Multi-peak shape optimisation results for a constrained inclined slotted hole with $b/r = 2$ in a uniaxially-loaded large square plate. (a) Initial and optimal shapes. (b) K_t around hole perimeter for initial and optimal shapes.*

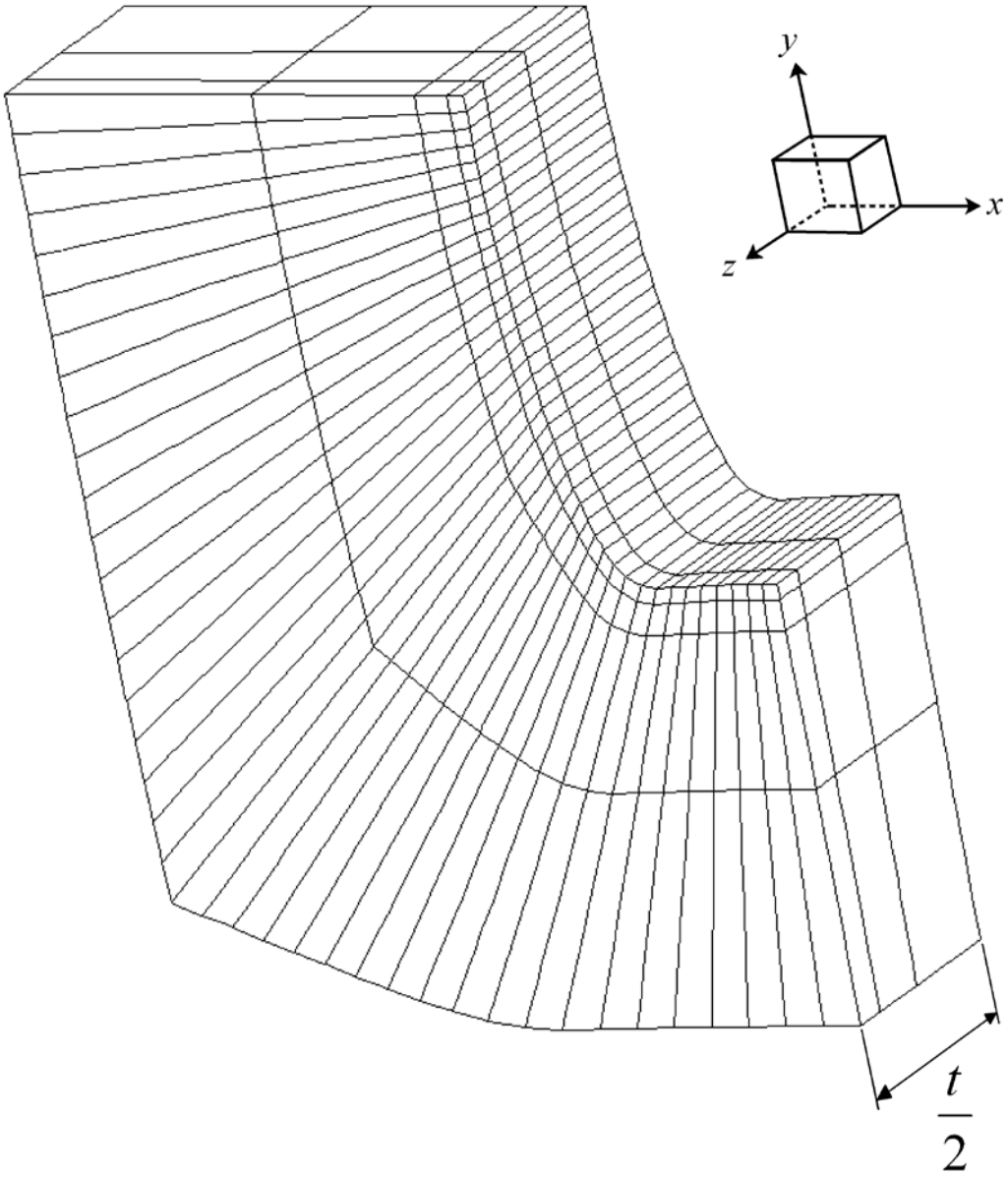
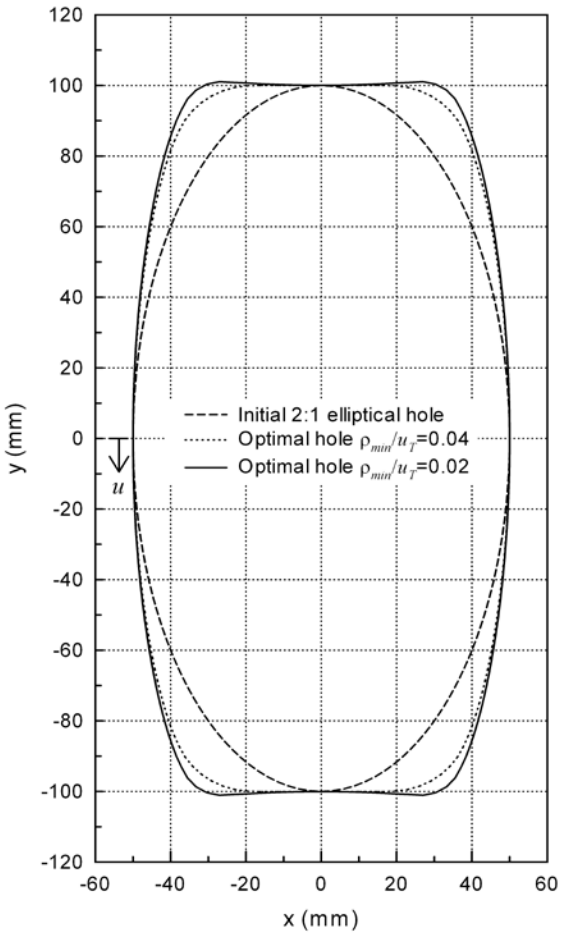
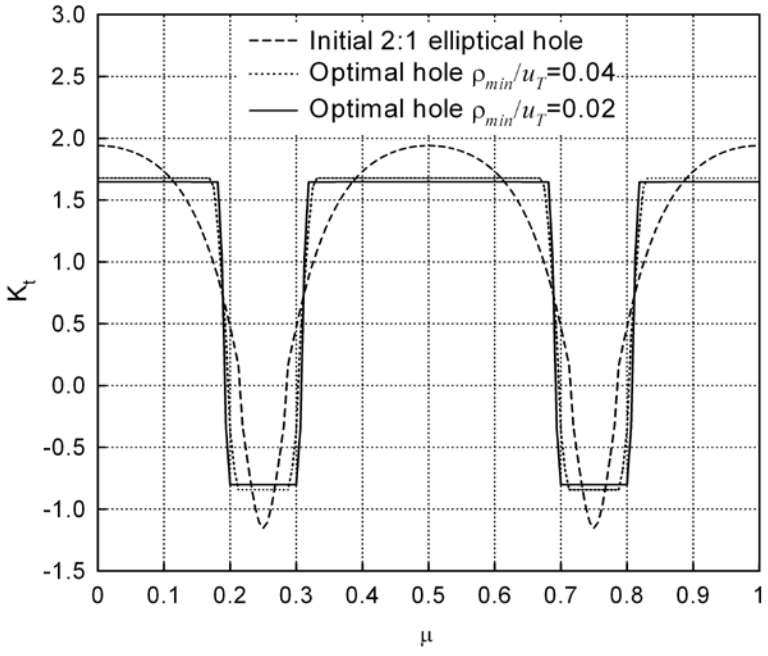


Figure 15: *Local detail of ⅛-symmetry FEA mesh for typical 3D optimal hole in a uniaxially-loaded large thick rectangular plate of thickness t (h:w = 2:1 and ρ_{min}/u_T = 0.02).*



(a)



(b)

Figure 16: *Multi-peak shape optimisation results for a 3D hole of near 2:1 aspect ratio in a uniaxially-loaded large thick plate. (a) Initial and optimal shapes. (b) Peak through-thickness K_t around hole perimeter for initial and optimal shapes.*

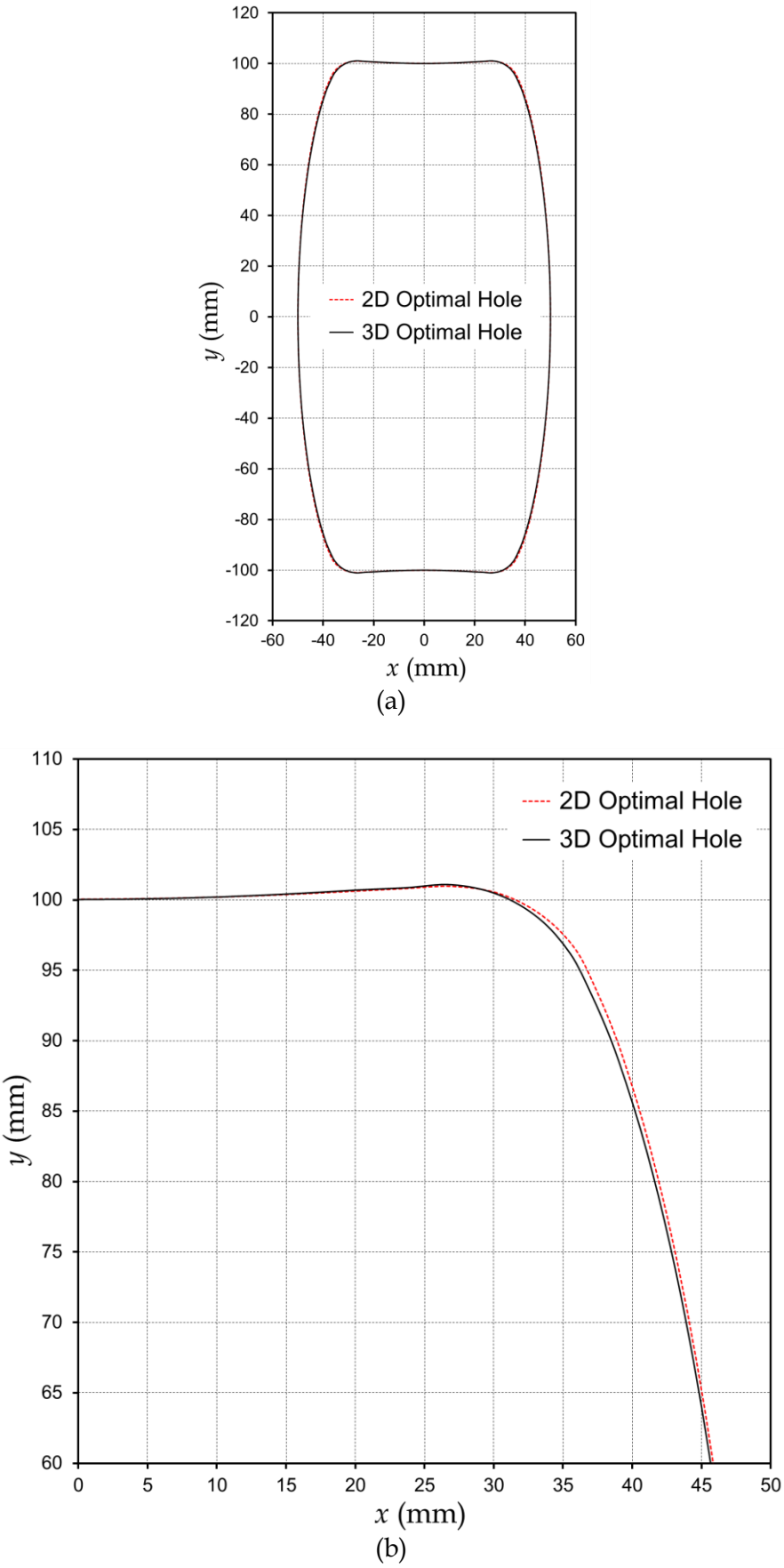


Figure 17: *Comparison of optimal shapes for a 2:1 hole in 2D and 3D uniaxially-loaded plates. a) Full shape. b) Close-up of shape in corner region.*

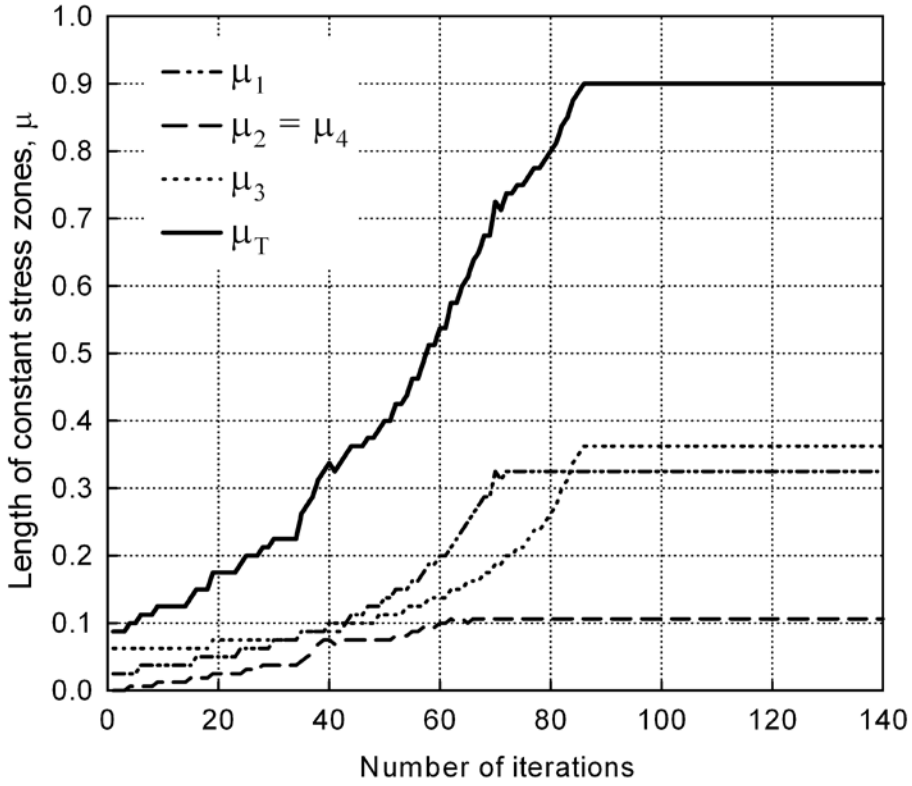


Figure 18: *Typical convergence of lengths of the constant stress zones for a 2:1 hole close to an edge in a large plate for* $\rho_{min}/u_T = 0.02$.

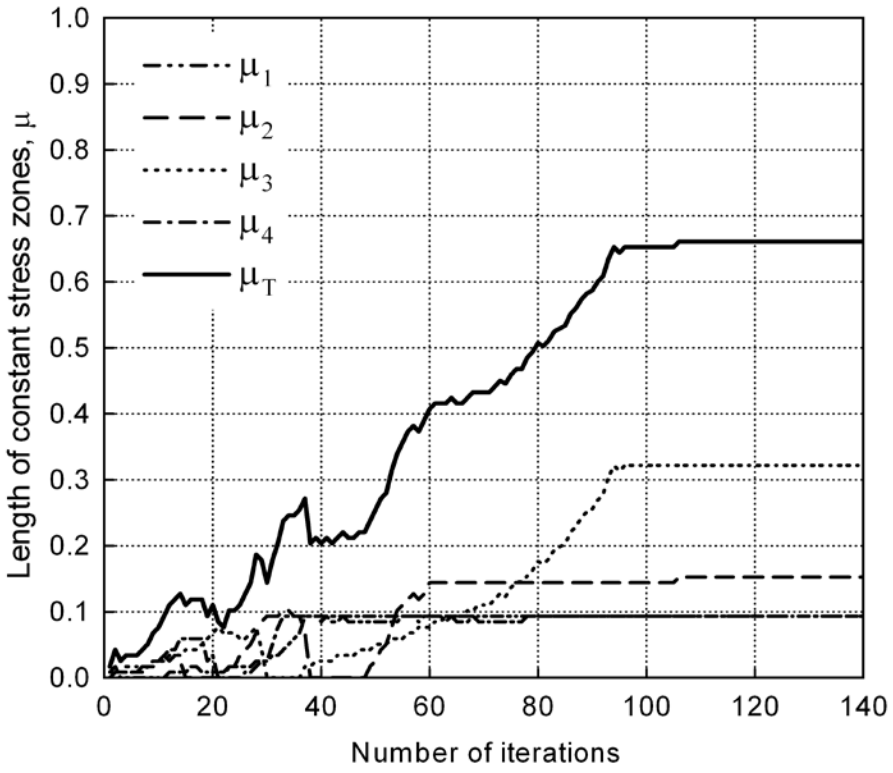


Figure 19: *Typical convergence of lengths of the constant stress zones for a 2:1 hole in a large plate where the hole is subjected to an inclined geometric constraint line and* $\rho_{min}/u_T = 0.02$.

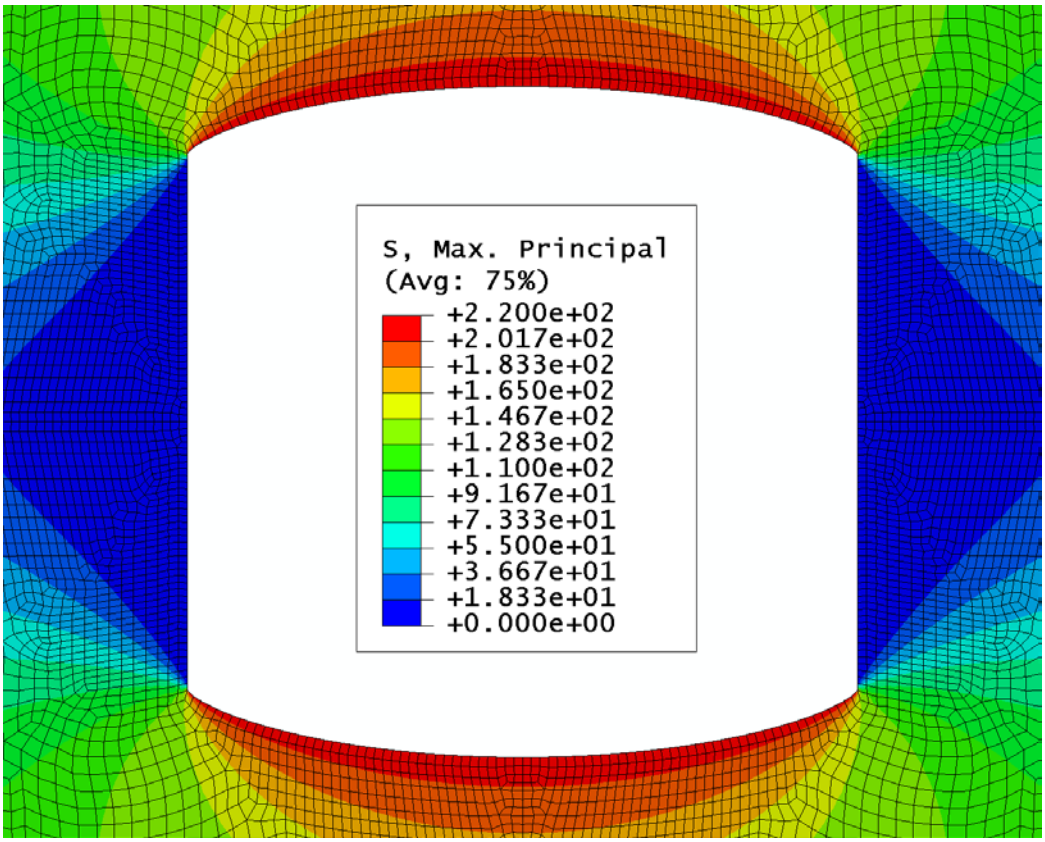


Figure 20: *Contours of maximum principal stress for a 1:1 optimal sharp-cornered uniaxially-loaded (S_2:S_1 = 1:0) hole shape. Geometry taken from Burchill and Heller (2004b).*

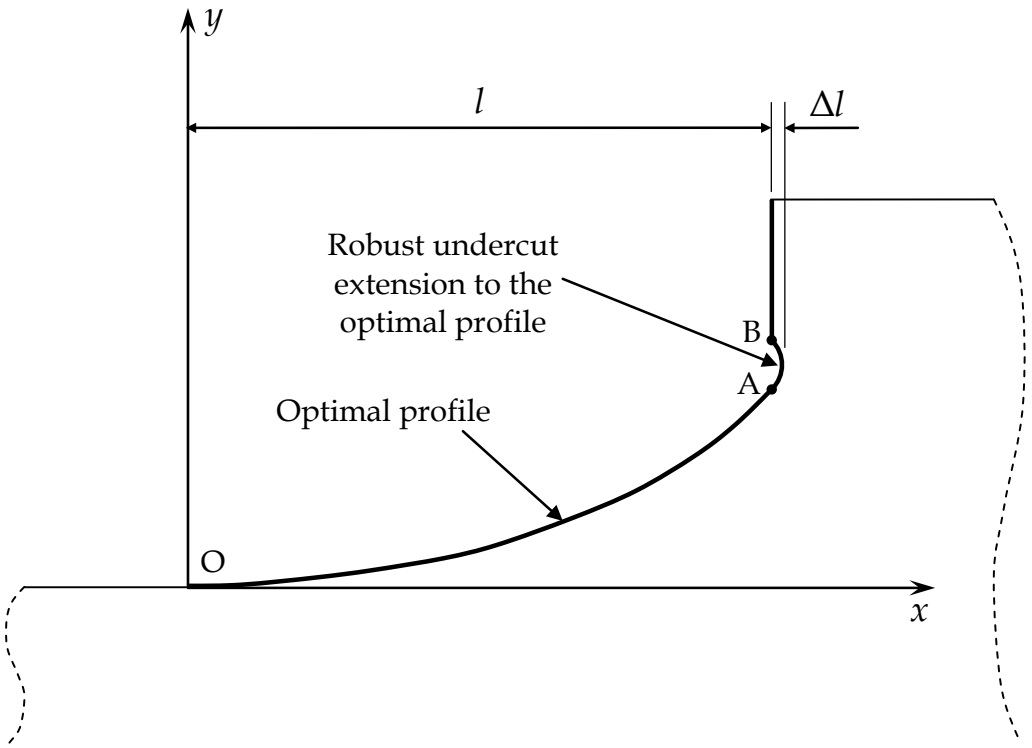


Figure 21: *Schematic for computation of a circular undercut robust extension for an sharp-cornered optimal hole profile.*

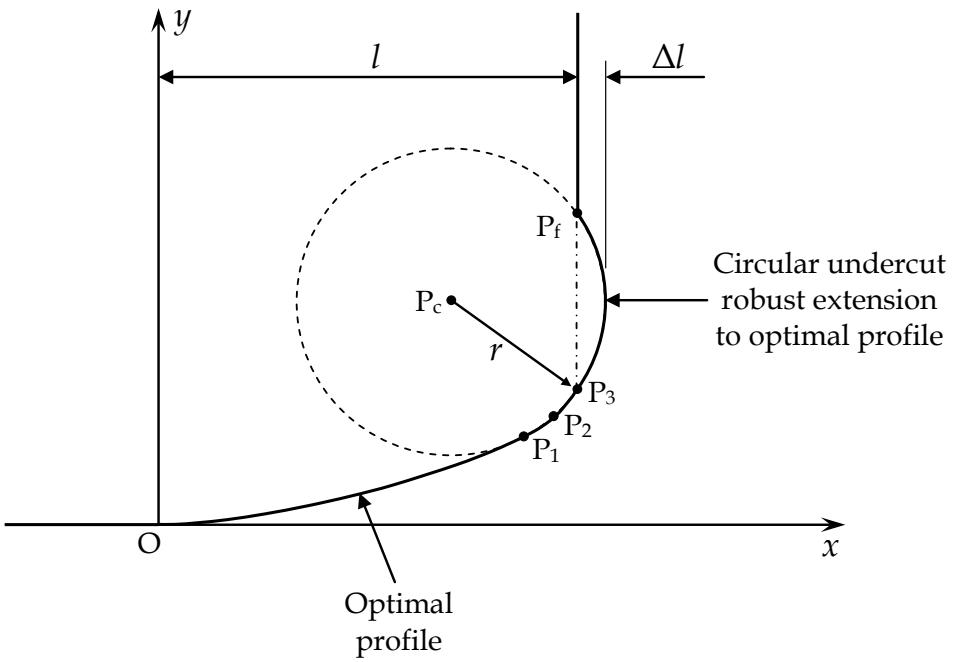


Figure 22: *Schematic for computation of a circular undercut robust extension for an sharp-cornered optimal hole profile.*

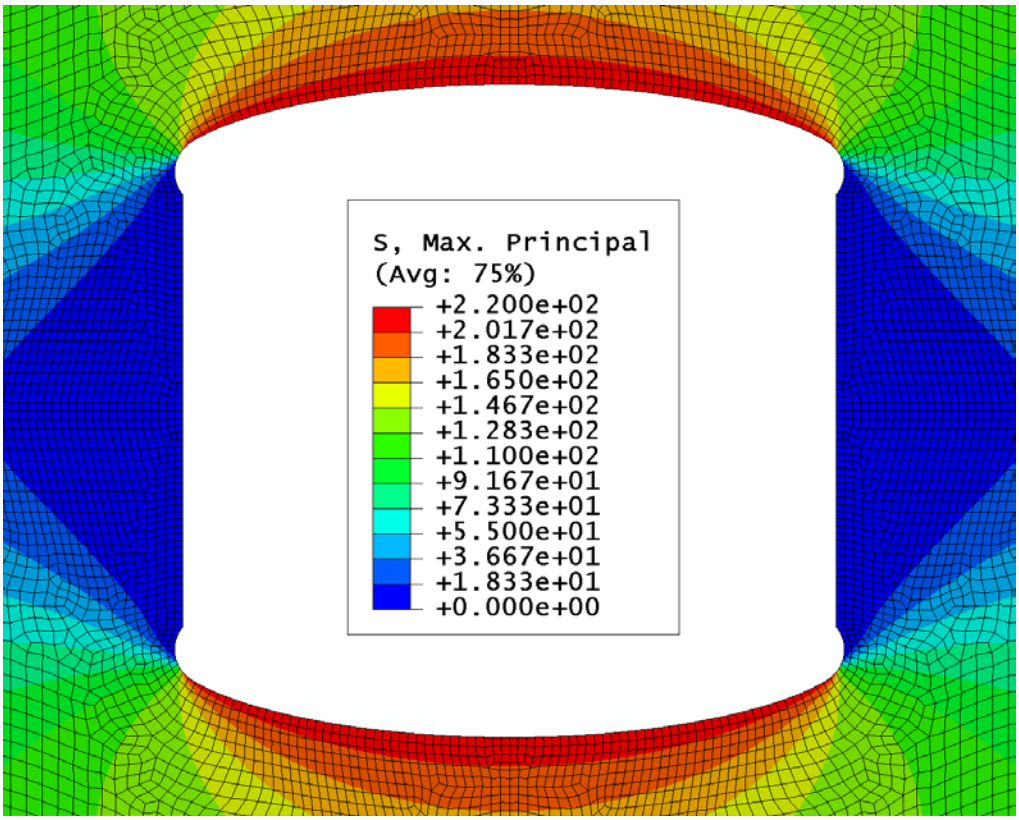


Figure 23: *Contours of maximum principal stress for a 1:1 robust optimal uniaxially-loaded (S_2:S_1 = 1:0) hole shape with circular undercuts added to the corner regions. Initial sharp-cornered geometry taken from Burchill and Heller (2004b).*

Appendix A:
FORTRAN 90 program for multi-peak shape optimisation using the PAFEC finite element analysis code

A.1. Example shape optimisation program control deck

The following text corresponds to an example shape optimisation control deck stored in the file called `MPUNI010AAOPT.DAT`. This particular job corresponds to the shape optimisation of a 1:1 hole in a large square plate under uniaxial loading. Note that the data lines associated with the NAMELIST variables `bndnodes`, `fixnodes`, `actnodes` and `mirnodes` have been split across two lines.

```
$INPUTS
  puppiesfile='/home/waldmanw/bin/puppies.exe'
  puppiesversion='@8.5'
  title1='Quarter-model of large square plate with 1:1 hole under uniaxial loading.'
  title2='r/h = 0.10 (r = min radius of curvature, h = initial hole height)'
  title3='PlateWidth/HoleDiameter = 10'
  r=100.0
  s=0.004
  c=0.001
  itmax=250
  nogrowth=.false.
  smoothdelta=.false.
  smoothxy=.false.
  maxsmooth=15
  optth=0
  optwt=0
  echotoscn=.false.
  setmidsidenodes=.false.
  chkconvergence=.false.
  chkdivergence=.false.
  constspacing=.true.
  minradcurvlim=10.0
  multiplepeaks=.true.
  restorebnd=.true.
  closedbnd=.false.
  pafecname='MPUNI010A'
  resultsname='MPUNI010AA'
  bndpolygon=(-100.0,70.0) (-100.0,-70.0) (100.0,-70.0) (100.0,70.0)
  bndnodes=001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020
 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041
  fixnodes=  2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
  actnodes=  T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T
  T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T   T
  mirnodes=  2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
$END
```

A.2. Example PAFEC data deck for use in shape optimisation

The following is a listing of the initial PAFEC input data deck stored in the file `MPUNI010A.DAT`. This data deck is subsequently modified at every iteration by the shape optimisation program, in order to define the new geometry prior to running the next PAFEC analysis. In this example, the compact and concise nature of a PAFEC data file is evident.

```
C 13-01-2003 Witold Waldman
C
C Data file for FE model of a large square plate with a 1:1 circular hole.
C
C PlateWidth/HoleDiam = 10.0
C
```

```
C Uniform tensile stress applied to top and bottom edges.
C
C Midside nodes of PAFBLOCKS around hole boundary defined
C to allow curved element edges (rather than straight sides).
C
CONTROL
FULL.CONTROL
TOLERANCE=10E-4
PHASE=1
PHASE=2
SKIP.CHECK
PHASE=4
PHASE=6
PHASE=7
PHASE=9
STOP
CONTROL.END
C
NODE
NODE.NUMBER   X           Y
    001   -50.000000     0.000000
    041     0.000000   -50.000000
C
    201  -500.000000     0.000000
    221  -500.000000  -500.000000
    241     0.000000  -500.000000
C
C Nodes along initial circular hole boundary.
C
      1       -50.000000              0.00000000
      2       -49.961448             -1.9629904
      3       -49.845867             -3.9229545
      4       -49.653419             -5.8768691
      5       -49.384418             -7.8217229
      6       -49.039264             -9.7545156
      7       -48.618495            -11.672268
      8       -48.122760            -13.572022
      9       -47.552825            -15.450848
     10       -46.909568            -17.305852
     11       -46.193977            -19.134170
     12       -45.407161            -20.932987
     13       -44.550327            -22.699525
     14       -43.624801            -24.431062
     15       -42.632007            -26.124927
     16       -41.573478            -27.778511
     17       -40.450847            -29.389261
     18       -39.265846            -30.954696
     19       -38.020298            -32.472400
     20       -36.716129            -33.940036
     21       -35.355339            -35.355339
     22       -33.940036            -36.716129
     23       -32.472400            -38.020298
     24       -30.954694            -39.265846
     25       -29.389259            -40.450851
     26       -27.778509            -41.573482
     27       -26.124925            -42.632007
     28       -24.431060            -43.624801
     29       -22.699525            -44.550327
     30       -20.932985            -45.407157
     31       -19.134170            -46.193977
     32       -17.305850            -46.909564
     33       -15.450848            -47.552825
     34       -13.572020            -48.122760
     35       -11.672265            -48.618495
     36        -9.7545128           -49.039264
     37        -7.8217191           -49.384418
     38        -5.8768672           -49.653419
     39        -3.9229545           -49.845867
     40        -1.9629906           -49.961452
     41         0.00000000          -50.000000
C
PAFBLOCKS
TYPE=1
ELEMENT.TYPE=36210
GROUP=1
PROPERTIES=1
  N1  N2     TOPOLOGY
```

```
  21  22     001 003 201 203 002
  21  22     003 005 203 205 004
  21  22     005 007 205 207 006
  21  22     007 009 207 209 008
  21  22     009 011 209 211 010
  21  22     011 013 211 213 012
  21  22     013 015 213 215 014
  21  22     015 017 215 217 016
  21  22     017 019 217 219 018
  21  22     019 021 219 221 020
  21  22     021 023 221 223 022
  21  22     023 025 223 225 024
  21  22     025 027 225 227 026
  21  22     027 029 227 229 028
  21  22     029 031 229 231 030
  21  22     031 033 231 233 032
  21  22     033 035 233 235 034
  21  22     035 037 235 237 036
  21  22     037 039 237 239 038
  21  22     039 041 239 241 040
C
C ARC.NODES
C LIST.OF.NODES.ON.ARC
C  001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020
C *021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040
C *041
C
LINE.NODES
LIST.OF.NODES.ON.LINE
  201 202 203 204 205 206 207 208 209 210
* 211 212 213 214 215 216 217 218 219 220 221
  221 222 223 224 225 226 227 228 229 230
* 231 232 233 234 235 236 237 238 239 240 241
C
MATERIAL
MATERIAL.NUMBER  E        NU
11               73000    0.32
C
PLATES.AND.SHELLS
PLATE.NUMBER MATERIAL.NUMBER THICKNESS
1            11              1.0
C
MESH
REFERENCE SPACING.LIST
21         2
22         1 1 1 1 2 4 8 12 16 20 24 28 32
C
RESTRAINTS
NODE PLANE AXIS DIRECTION
001  2     1    2
041  1     1    1
C
SURFACE.FOR.PRESSURE
PRESSURE.VALUE     NODE PLANE
  -100.0           241  2
C
END.OF.DATA
```

A.3. Source code listing of shape optimisation program

The following text corresponds to the FORTRAN 90 source code listing of the most recent version of the multi-peak shape optimisation program. The name of the file is `optpafec042.f90`. This version of the program was developed to run under the HP-UX 11.0 operating system on a computer called `jayted`.

```
!=============================================================================================

      program optpafec042

! Author: Witold Waldman
! Date  : 22-05-1998
!
! This program moves nodes on a boundary during a PAFEC structural shape optimisation run.
```

```
!
! The (long) command line used on the HP computer to create the executable from this
! FORTRAN 90 source code is as follows:
!
!       f90 +U77 +O0 -g +check=all -o optpafec042.exe  \
!         ~/library/fparser/parameters.f90             \
!         ~/library/fparser/fparser.f90                \
!         ~/library/fitpack/dfitpack.f90               \
!         ~/library/fitpack/arczerocurv.f90            \
!         ~/library/locpt/locpt.f90                    \
!         optpafec042.f90
!
! The +U77 option invokes support for the BSD 3F library, some of whose routines are used.
!
! Alternatively, the HP-UX make utility can be invoked with the default make file
! called "makefile". For example:
!
!       make
!
! Otherwise, the make file can be specified explicitly on the command line:
!
!       make -f makefile
!
! 25-03-1998
!
! Added code for specifying x and y limits that should not be crossed during an optimisation run.
!
! 07-04-1998
!
! Added code to test for convergence taking into account that some nodes may have reached
! the user-defined move limits.
!
! 21-05-1998
!
! Modified comment printed with iteration number to make it clear that the nodal coordinates
! are the predicted ones.
!
! 22-05-1998
!
! Added code to allow the PAFEC jobs to be executed from within this program during the
! optimisation process.
!
! 02-12-1998
!
! Added code to allow the parameters for the optimisation job to be read from a data file.
! The NAMELIST feature of FORTRAN 77/90 is used to accomplish this with a minimum of coding
! related to data parsing.
!
! 17-02-1999
!
! Added code to test for divergence of optimisation solution.
!
! 25-02-1999
!
! Added code to test for a converged solution by checking the last few values of maximum
! boundary stress to see if they are identical.
!
! 30-07-1999
!
! Reorganised the code to compute the angle of the outward normal and then use this directly.
!
! 03-08-1999
!
! Added code to write out information that helps to identify the optimal solution. This is
! useful when divergence occurs, or the solution is oscillating without a definite convergence.
!
! 22-09-1999
!
! Corrected equation for calculating nodal movement to use the stress threshold rather than
! the average boundary stress. This matches the algorithm given in the Kaye and Heller shape
! optimisation report.
!
! 23-09-1999
!
! Modified algorithm to use absolute values of the boundary stress when calculating nodal
! movements. This enables regions with compressive stresses to be optimised by the same approach.
!
```

```
! 05-10-1999
!
! Modified the nodal movement algorithm to choose for the threshold stress the stress that
! has the largest absolute value.
!
! 06-10-1999
!
! Modified algorithm to use the average of the absolute values of the boundary stresses when
! the stress threshold is being set to equal an average stress. Provision made for the user to
! exclude the a given number of the lowest absolute boundary stress values from the calculation
! of the average.
!
! 11-10-1999
!
! Added code to enforce rework around a circular boundary.
!
! 05-01-2000
!
! Increased by one the number of decimal places used to write out the nodal coordinates. This
! was done to allow finer movements to be defined during the optimisation when close to the
! true optimal solution.
!
! 06-01-2000
!
! Added code to ensure that midside nodes of PAFBLOCKs are exactly placed on the midside of
! the arc or line defining the edge.
!
! 25-01-2000
!
! Modified writing of results so that they are now sent to two files. The larger file,
! containing the stress results at the nodes for each iteration, is now appended to at
! each iteration. This will save considerable file I/O, particularly for jobs where
! hundreds of iterations are necessary.
!
! 18-02-2000
!
! Added checks to see that node coordinates are only written when the first PAFBLOCKS
! statement is encountered.
!
! 22-03-2000
!
! Made provision for the user to supply up to three lines of text to be used as a title near
! the beginning of some of the output files.
!
! 30-03-2000
!
! Added check to see if this job is being run as a restart directly from a previous optimisation
! job. The comment line associated with the iteration number is edited to prevent it from causing
! the nodal updating procedure from working erroneously.
!
! 16-10-2000
!
! Added code to allow user to specify the PUPPIES executable file that is to be used for
! extracting stress data from a PAFEC finite element analysis run.
!
! 14-11-2000
!
! Added code to compute the radius of curvature at each node.
!
! 24-11-2000
!
! Added code to keep the spacing of the nodes along moving boundary constant, so that mesh
! distortion might be avoided.
!
! 19-12-2000
!
! Added code to modify optimisation process to use the local radius of curvature to alter
! the amount of material being removed.
!
! 16-01-2001
!
! Added code to enable smoothing to be switched on or off for nodes along the moving boundary.
!
! 25-01-2001
!
! Added code to restrict the nodal movements in regions where the minimum radius of curvature
! requirement is violated.
```

```
!
! 07-02-2001
!
! Added code to allow the stress threshold to be set equal to the mean of the maximum absolute
! boundary stress and the minimum absolute boundary stress.
!
! 01-03-2001
!
! Added code to identify locations of corners and to adjust the mesh spacing on adjacent
! faces to prevent mesh distortion from occuring.
!
! 03-08-2001
!
! Modified program to enable quasi-3D shape optimisation to be performed.
!
! 27-09-2001
!
! Added code to enable von Mises stress to be used for the optimisation. This stress is
! directly available in the PUPPIES output file.
!
! 04-06-2002
!
! Removed code that alters the boundary stress value based on the minimum radius of curvature
! (that was used to compute nodal movements). This code is redundant with the boundary correction
! based on the local radius of curvature. Most of the time the removed code was not being
! activated, as the radius of curvature only went slightly below the minimum allowed.
!
! 04-07-2002
!
! Modified the code that applies the radius of curvature constraint to enable fixing of the X
! and/or Y coordinates. This capability was not included in the original implementation.
!
! 24-10-2002
!
! Added code that would allow a bounding polygon to be defined for use during a constrained
! shape optimisation run.
!
! 01-11-2002
!
! Added code to allow the stress threshold to be set based on the stress at a particular
! user-defined node.
!
! 16-04-2003
!
! Added code to allow the stress threshold associated with a particular peak to be modified
! for the first iteration using a user-supplied value.
!
! 19-05-2003
!
! Added code to compute and display the arc length around the moving boundary.
!
! 18-08-2003
!
! Added code to convert old rectangular bounding box data input to the new bounding
! polygon format. This improves compatibility with older data files.
!
! 20-08-2003 Version 4.1.001
!
! Fixed code that computes the lengths of the regions of uniform stress to correctly handle
! cases with a closed boundary. The problem only affected results where the uniform region
! at the end of the list of nodes was a continuation of the region at the start of the list
! of nodes (i.e. the starting and ending regions of uniform stress were part of one
! continuous zone of uniform stress).
!
! 30-10-2003 Version 4.2.001
!
! Modified code to output stress data for multiple stress peaks in the summary file.
! Incremented minor version number as changes to the output format were required.
! Deleted from the output files some statistics that were not all that useful and
! were cluttering up the output listing. Added additional statistics related to the
! stress behaviour in each zone.
!
! 28-11-2003 Version 4.2.002
!
! Added facility for user-defined equations to transform the (x,y,z) coordinates.
! Also, provided option for certain transform equations requiring "if" statements
! to be hard-coded, with the facility to include user-supplied parameters to allow
```

```
! different cases to be analysed.
!
! 04-12-2003
!
! Added facility to use the maximum positive peak for the stress threshold in regions
! where the stresses are positive.
!
! 23-12-2003 Version 4.2.003
!
! Modified code so that it does not include the absolute value signs in the formula
! used to compute the nodal movements, unless specifically selected by a user option.
!
! 12-01-2004 Version 4.2.004
!
! Added code to output the iteration at which the maximum length of constant stress
! boundary first occurs. This will enable the best solution to be identified more
! reliably. Simply looking for the minimum peak stress or minimum stress range
! (between positive and negative peaks) is not always the best indicator, as
! the constant-stress regions may not not have had time to develop fully.
!
! 28-01-2004 Version 4.2.005
!
! Added code to allow a factor to be applied to the boundary length, which is
! useful when dealing with models that have quarter or half symmetry.
!
! 13-02-2004 Version 4.2.006
!
! Modified code to reduce maximum number of passes through the radius of
! curvature modification code (600 was way too many). Also adjusted the
! code to use original coordinates prior to nodal movement being applied
! if the maximum allowable number of passes has been achieved.
!
! 01-03-2004 Version 4.2.007
!
! Added code to read in a supplementary list of nodes that are to be used when
! performing customised processing for particular models.
!
! 03-06-2004 Version 4.2.008
!
! Added code to output the maximum of the three principal stresses.
! Changed output column heading from Poly to Insd to better represent
! that this item denotes whether the node lies inside, on, or outside
! the polygonal boundary.
!
! 22-07-2004 Version 4.2.009
!
! Corrected indexing error in code that selects the maximum boundary
! stress (affects cases where we have more than one boundary).

      use parameters, only: rn
      use fparser,    only: initf, parsef, evalf, EvalErrType, EvalErrMsg

      implicit none

      integer, parameter :: lui=1        ! Logical unit for input data file.
      integer, parameter :: lup=3        ! Logical unit for PUPPIES file.
      integer, parameter :: luj=4        ! Logical unit for job data file.
      integer, parameter :: lut=7        ! Logical unit for temporary file.
      integer, parameter :: luo=8        ! Logical unit for job out file.
      integer, parameter :: maxit=600    ! Max number of iterations.
      integer, parameter :: maxnode=170  ! Max number of nodes to optimise.
      integer, parameter :: maxdivg=10   ! Max number of diverging iterations.
      integer, parameter :: maxbnd=5     ! Max number of boundaries.
      integer, parameter :: maxzone=10   ! Max number of stress zones along boundary.
      integer, parameter :: maxpoly=20   ! Max number of points defining polygon.

      logical    solved,nogrowth,smoothdelta,smoothxy,restart
      logical    chkxbnd,chkybnd,echotoscn,setmidsidenodes
      logical    jobdiverging,chkconvergence,chkdivergence
      logical    actnodes(maxnode),xfixnodes(maxnode),yfixnodes(maxnode)
      logical    xmirnodes(maxnode),ymirnodes(maxnode)
      logical    constspacing,closedbnd
      character  fixcode*8,mircode*8
      character  scn*256
      character  aline*80,pnpfin*80
      character  opdfile*256
      character  resultsfile*256
```

```
character  resultsname*256
character  pafecfile*256
character  pafecname*256
character  joboptoutfile*256
character  joboptsmyfile*256
character  joboptpfcfile*256
character  joboptscnfile*256
character  pupjnlfile*256
character  pupoutfile*256
character  bndnodesfile*256
character  tempfile*256
character  results09file*256
character  title1*128,title2*128,title3*128
character  puppiesfile*256
character  datestamp*24
character  programid*80
character  maxitmsg*64
character  runfailedmsg*64
character  puppiesversion*5
integer    i,j,k,l,m,ios,itnum,ntemp,maxsmooth,optth,optwt,itmax
integer    nclarg,nactnodes,time0,iact,nnodestot,ncorners,itcorner
integer    ielem
logical    init,issamepoint,pafblocksfound,elementsonright
logical    restorebnd,findcorners,errorinxyz,is2dgeom
logical    iscorner(maxnode),stopafterpuppies,invalidoption
logical    useglobalstress,useaveragedstress,usevonmisesstress
real*8     rnum,bndstrtot,vartot,maxdist(maxit)
real*8     c,r,s,angle,theta(maxnode,maxit),sr,pi,pion2,twopi,wt
real*8     xbndlft,xbndrgt,ybndtop,ybndbot,elapsedtime,dt
integer    np(maxnode,maxbnd,maxit),npsup(maxnode,maxbnd,maxit)
integer    nbndnodes,nbnd,bndnodes(maxnode,maxbnd),fixnodes(maxnode)
integer    nsupnodes,nsupbnd,supnodes(maxnode,maxbnd)
integer    ipsupnode(maxnode),jpsupnode(maxnode)
integer    mirnodes(maxnode),clockwise(maxnode),smthnodes(maxnode)
integer    concave(maxnode,maxit),concavenew(maxnode)
integer    icorners(maxnode),nnodeedge(maxnode)
integer    iedgebeg(maxnode),iedgeend(maxnode)
real*8     xpsup(maxnode,maxbnd,maxit),ypsup(maxnode,maxbnd,maxit),zpsup(maxnode,maxbnd,maxit)
real*8     xp(maxnode,maxbnd,maxit),yp(maxnode,maxbnd,maxit),zp(maxnode,maxbnd,maxit)
real*8     xr(maxnode,maxbnd,maxit),yr(maxnode,maxbnd,maxit),zr(maxnode,maxbnd,maxit)
real*8     xn(maxnode,maxbnd,maxit),yn(maxnode,maxbnd,maxit),zn(maxnode,maxbnd,maxit)
real*8     xd(maxnode,maxit),yd(maxnode,maxit)
real*8     xs(maxnode),ys(maxnode)
real*8     xu(maxnode),yu(maxnode)
real*8     bndstr(maxnode,maxit)
real*8     dr(maxnode,maxit),drs(maxnode),da(maxnode,maxit)
real*8     s1(maxnode,maxbnd,maxit),s2(maxnode,maxbnd,maxit),s3(maxnode,maxbnd,maxit)
real*8     bigs1s2s3(maxnode,maxbnd,maxit),svm(maxnode,maxbnd,maxit)
real*8     s11,s22,s33,a11,a22,a33
real*8     sxx,syy,szz,txy,tyz,tzx
real*8     snodei,snodeip1
real*8     deltabndstr(maxnode,maxit)
real*8     avgbndstr(maxit),sth(maxzone,maxit)
real*8     strthreshfac(maxzone)
real*8     zonepeak(maxzone,maxit),zonet(maxzone,maxit),zonesd(maxzone,maxit)
real*8     zoneavg(maxzone,maxit),zonerange(maxzone,maxit),zonemin(maxzone,maxit)
real*8     zonetotlen(maxit),pcinit(maxit)
real*8     maxbndstr(maxit),minbndstr(maxit),rngbndstr(maxit),dltbndstr(maxit)
real*8     maxabsbndstr(maxit),minabsbndstr(maxit),avgabsbndstr(maxit)
real*8     sortabsbndstr(maxnode),actbndstr(maxnode)
real*8     rngabsbndstr(maxit),dltrngbndstr(maxit)
real*8     minrngbndstr,minmaxbndstr,minabsbndstrit,maxzonetotlen
real*8     minrngabsbndstr,anglecorner,maxonbndj,maxsvm,radtodeg
real*8     radius,reworkcircle(3),rwkclrad,rwkclxc,rwkclyc
real*8     xint1,yint1,zint1,xint2,yint2,zint2,dst1,dst2,stol
real*8     minradcurv(maxit),maxradcurv(maxit),avgradcurv(maxit)
real*8     radcurv(maxnode,maxit)
real*8     radcurvnew(maxnode),inclangle(maxnode)
real*8     minradcurvlim,minrc,sbndtotal,scurbndtotal
real*8     tbnd(maxnode),sbnd(maxnode),tempkurv(maxnode)
real*8     tcurbnd(maxnode),scurbnd(maxnode)
real*8     sigmakurv,slope1kurv,slopenkurv
real*8     xpkurv(maxnode),ypkurv(maxnode),skurv(maxnode)
real*8     sigmakurvp,tempkurvp(2*maxnode)
real*8     xpkurvp(maxnode),ypkurvp(maxnode),skurvp(maxnode)
real*8     xnew(maxnode),ynew(maxnode)
integer    iminrngbndstr,iminmaxbndstr,imaxzonetotlen
```

```
integer    iminabsbndstrit,iminrngabsbndstr,ipos,nodeth,inodeth
integer    nskiplowabs,ni,isfkurv,ierrkurv,ierrkurvp,nrcpass,maxrcpass
integer    nzonesmax
integer    nzc,izcbeg(maxzone),izcend(maxzone)
integer    nzones(maxit),izone(maxnode),izonepeak(maxzone)
logical    closedbnd,multiplepeaks,custompuppies,rectbnd
logical    wasconstrained,allpointsinside,xpypinside(maxnode)
integer    npolygon,lp,m
real*8     xbndpolygon(maxpoly),ybndpolygon(maxpoly)
complex*16 bndpolygon(maxpoly)
logical    usexyzeqn
character  xeqn*256,yeqn*256,zeqn*256
character  xyzeqn(3)*256
character  xyzeqnvar(3)*2
real(rn)   xyzeqnval(3)
integer    optmodifyxyz,nuserparam
real*8     userparam(20),xo,yo,zo
real*8     b,ts,rf,af
logical    usemaxpospeaksth
real*8     maxpospeak
logical    useabsvaluesdr
real*8     bndlenfac
logical    skipstressextraction

real*8     davgvec,biggestof2,biggestof3,dsumvec,dmaxvec
integer    iargc,time,imaxvec
logical    fexist,within,inrange
character  fdate*24,ZHstr*11

namelist /inputs/ r,s,c,itmax,nogrowth,smoothdelta,smoothxy,      &
                  maxsmooth,optth,optwt,chkybnd,chkxbnd,          &
                  xbndlft,xbndrgt,ybndtop,ybndbot,custompuppies,  &
                  resultsname,pupjnlfile,pafecname,bndnodesfile,  &
                  tempfile,bndnodes,echotoscn,chkdivergence,      &
                  fixnodes,actnodes,mirnodes,restart,nskiplowabs, &
                  reworkcircle,chkconvergence,setmidsidenodes,    &
                  title1,title2,title3,puppiesfile,               &
                  minradcurvlim,constspacing,elementsonright,     &
                  restorebnd,smthnodes,findcorners,itcorner,      &
                  anglecorner,stopafterpuppies,puppiesversion,    &
                  useglobalstress,useaveragedstress,              &
                  usevonmisesstress,closedbnd,multiplepeaks,      &
                  bndpolygon,nodeth,strthreshfac,stol,xeqn,yeqn,  &
                  zeqn,optmodifyxyz,userparam,usemaxpospeaksth,   &
                  useabsvaluesdr,bndlenfac,supnodes

programid='======= OptPAFEC Version 4.2.009 ======='

nclarg=iargc()

if (nclarg == 0 .or. nclarg > 1) then
  write(*,*)
  write(*,*) programid(1:len_trim(programid))
  write(*,*)
  write(*,*) 'This program is used to perform shape optimisation.'
  write(*,*)
  write(*,*) 'Usage: optpafec042.exe optjob.dat'
  write(*,*)
  stop
endif

write(*,*)

datestamp=fdate()
datestamp=datestamp(1:4)//datestamp(9:11)//datestamp(5:8)//datestamp(21:24)//datestamp(11:19)
if (datestamp(5:5) == ' ') datestamp(5:5)='0'
time0=time()

! Calculate some math constants.

pi=4.0d0*atan(1.0d0)
pion2=pi/2.0d0
twopi=2.0d0*pi
radtodeg=180.0d0/pi

! Set up default values of iteration job parameters.
```

```
r=100.0d0                          ! Characteristic length.
s=0.025d0                          ! Step size scaling factor.
c=0.002d0                          ! Convergence criterion.
itmax=200
nogrowth=.false.
smoothdelta=.false.
smoothxy=.false.
restart=.false.
maxsmooth=0
optth=0
optwt=0
stol=0.01d0
chkybnd=.false.
chkxbnd=.false.
pafecname='SPARE'
resultsname='JOB'
pupjnlfile='PUP.JNL'
bndnodesfile='BND.NODES'
tempfile='TEMP.DAT'
echotoscn=.true.
xbndlft=0.0d0
xbndrgt=0.0d0
ybndtop=0.0d0
ybndbot=0.0d0
nbndnodes=0
nactnodes=0
nbnd=1
nsupbnd=0
nsupnodes=0
do i=1,maxnode
  do j=1,maxbnd
    bndnodes(i,j)=-1
    supnodes(i,j)=-1
  enddo
  fixnodes(i)=0
  smthnodes(i)=0
  actnodes(i)=.true.
enddo
do i=1,3
  reworkcircle(i)=0.0d0
enddo
chkdivergence=.false.
chkconvergence=.false.
setmidsidenodes=.false.
nskiplowabs=0
title1=''
title2=''
title3=''
puppiesfile=''
custompuppies=.false.
minradcurvlim=20.0d0
constspacing=.false.
elementsonright=.true.
restorebnd=.false.
findcorners=.false.
itcorner=0
anglecorner=110.0d0
stopafterpuppies=.false.
useglobalstress=.false.
useaveragedstress=.false.
usevonmisesstress=.false.
puppiesversion='@8.14'
closedbnd=.false.
multiplepeaks=.false.
nodeth=0
do i=1,maxpoly
  bndpolygon(i)=(1.0d200,1.0d200)
enddo
do i=1,maxzone
  strthreshfac(i)=1.0d0
enddo
xeqn=' '
yeqn=' '
zeqn=' '
do i=1,3
  xyzeqn(i)=' '
enddo
```

```
optmodifyxyz=0
do i=1,size(userparam)
  userparam(i)=1.0d200
enddo
usemaxpospeaksth=.false.
useabsvaluesdr=.false.
bndlenfac=1.0d0

! Read in the optimisation control data from an input file.

call getarg(2,opdfile)   ! For f90 under HP-UX 10.2, need 2 not 1.

if (len_trim(opdfile) == 0) then
  ! If the string is empty, assume that filename is stored
  ! as the first argument under this operating system version.
  ! This makes the code run under HP-UX 11.0.
  call getarg(1,opdfile)
endif

if (.not. fexist(opdfile)) then
  write(*,'(1x,a,a,/)') '*** File not found: ',opdfile(1:len_trim(opdfile))
  stop
else
  write(*,'(1x,a,a)') '*** Optimisation data file: ',opdfile(1:len_trim(opdfile))
endif

open(unit=lui,file=opdfile)
read(unit=lui,nml=inputs,iostat=ios)
close(unit=lui)

if (ios == 0) then
  write(*,'(1x,a)') '*** Successfully read in set of optimisation data.'
  if (custompuppies) then
    write(*,'(1x,a  )') '*** The custompuppies option is no longer supported.'
    write(*,'(1x,a  )') '*** Use the puppiesfile and puppiesversion options instead.'
    write(*,'(1x,a  )') '***   e.g. puppiesfile=''/home/userid/bin/puppies.exe'''
    write(*,'(1x,a,/)') '***        puppiesversion=''@8.5'''
    stop
  endif
else if (ios > 0) then
  write(*,'(1x,a  )') '*** Error encountered while reading optimisation data.'
  write(*,'(1x,a,/)') '*** Check that all the supplied options are valid.'
  stop
else if (ios < 0) then
  write(*,'(1x,a,/)') '*** End-of-file reached while reading optimisation data.'
  stop
endif

! Check for options that are no longer supported.

if (chkxbnd .or. chkybnd) then
  write(*,*) '*** Warning: The chkxbnd and chkybnd options are no longer available.'
  write(*,*) '***          A polygon is now used to define geometric constraints.'
endif

! Check that the maximum number of iterations specified by the user is permissible.

if (itmax < 1 .or. itmax > maxit) then
  write(*,*) '***'
  write(*,*) '*** Maximum iterations should be in the range 1 to ',maxit,'.'
  write(*,*) '*** User-specified itmax = ',itmax
  write(*,*) '***'
  stop
endif

! Determine the number of boundary nodes. Here we assume that the number
! of nodes on each boundary (if there is more than one) is the same as
! that which is defined for the first boundary.

nbndnodes=0
do i=1,maxnode
  if (bndnodes(i,1) >= 0) nbndnodes=nbndnodes+1
enddo

! If it is defined, check that the node for calculation of the stress
! threshold exists in the user provided list of boundary nodes.
```

```
if (nodeth > 0) then
  inodeth=0
  do i=1,nbndnodes
    if (bndnodes(i,1) == nodeth) inodeth=i
  enddo
  if (inodeth == 0) then
    write(*,*) '*** Error: Node ',nodeth,' for stress threshold not found.'
    stop
  endif
endif

! Check validity of number of boundary nodes to ensure that
! the list of nodes has not been inadvertently truncated when
! it was read in.

if (nbndnodes < 4) then
  write(*,*) '***'
  write(*,*) '*** Minimum number of boundary nodes is 4.'
  write(*,*) '*** Number of boundary nodes = ',nbndnodes
  write(*,*) '***'
  stop
else if (nbndnodes > maxnode-1) then
  write(*,*) '***'
  write(*,*) '*** Maximum number of boundary nodes is ',maxnode-1,'.'
  write(*,*) '*** Number of boundary nodes = ',nbndnodes
  write(*,*) '***'
  stop
endif

! Determine the number of active nodes.

nactnodes=0
do i=1,nbndnodes
  if (actnodes(i)) nactnodes=nactnodes+1
enddo

if (nactnodes == 0) then
  write(*,*) '***'
  write(*,*) '*** The number of active nodes is zero.'
  write(*,*) '*** The optimisation cannot proceed.'
  write(*,*) '***'
  stop
endif

! Determine the number of boundary lines.

nbnd=0
do j=1,maxbnd
  if (bndnodes(1,j) > 0) nbnd=nbnd+1
enddo

if (nbnd == 1) then
  write(*,*) '*** Detected one set of boundary nodes.'
else
  write(*,*) '*** Detected',nbnd,'sets of boundary nodes.'
  if (nbnd == 0) stop
  if (useglobalstress.and.nbnd > 1) then
    write(*,*) '***'
    write(*,*) '*** Error: useglobalstress option valid only for use with a single boundary.'
    write(*,*) '***'
    stop
  endif
endif

! Check that the correct number of nodes were used to define each boundary line.

do j=1,nbnd
  do i=1,nbndnodes
    if (bndnodes(i,j) <= 0) then
      write(*,*) '***'
      write(*,*) '*** Error in node list for boundary line ',j,'.'
      write(*,*) '***'
      stop
    endif
  enddo
enddo
```

```
! Determine the number of supplementary nodes. Here we assume that the number
! of supplementary nodes on each boundary (if there is more than one) is the
! same as that which is defined for the first boundary.

nsupnodes=0
do i=1,maxnode
  if (supnodes(i,1) >= 0) nsupnodes=nsupnodes+1
enddo

! Determine the number of supplementary boundary lines.

nsupbnd=0
do j=1,maxbnd
  if (supnodes(1,j) > 0) nsupbnd=nsupbnd+1
enddo

if (nsupbnd > 0) then
  if (nsupbnd == 1) then
    write(*,*) '*** Detected one set of supplementary nodes.'
  else
    write(*,*) '*** Detected',nsupbnd,'rows of supplementary nodes.'
  endif
  write(*,*) '*** Total of',nsupnodes,'supplementary nodes detected.'
endif

! Check that the correct number of nodes were used to define
! each line of supplementary nodes.

do j=1,nsupbnd
  do i=1,nsupnodes
    if (supnodes(i,j) <= 0) then
      write(*,*) '***'
      write(*,*) '*** Error in node list for supplementary nodes line ',j,'.'
      write(*,*) '***'
      stop
    endif
  enddo
enddo

! Determine pointers to the (i,j) location of each of the master nodes.

do k=1,nsupnodes
  ipsupnode(k)=0
  jpsupnode(k)=0
  do j=1,nbnd
    do i=1,nbndnodes
      if (supnodes(k,1) == bndnodes(i,j)) then
        ipsupnode(k)=i
        jpsupnode(k)=j
      endif
    enddo
  enddo
enddo

! Determine the restraints on X and Y movement.

do i=1,nbndnodes
  xfixnodes(i)=.false.
  yfixnodes(i)=.false.
  write(fixcode,'(i8)') fixnodes(i)
  if (index(fixcode,'1') > 0) xfixnodes(i)=.true.
  if (index(fixcode,'2') > 0) yfixnodes(i)=.true.
enddo

! Check to see if a bounding rectangle has been defined using the
! old input data format.

rectbnd=.not. (xbndlft == 0.0d0 .and. xbndrgt == 0.0d0 .and.  &
               ybndtop == 0.0d0 .and. ybndbot == 0.0d0      )

! If a bounding polygon has been defined, determine how many points
! have been used to define its boundary.

npolygon=0
do i=1,maxpoly
  if (bndpolygon(i) /= (1.0d200,1.0d200)) then
    npolygon=npolygon+1
```

```
  else
    exit
  endif
enddo

! If user-defined parameters have been provided, determine how many.

nuserparam=0
do i=1,size(userparam)
  if (userparam(i) /= 1.0d200) then
    nuserparam=nuserparam+1
  else
    exit
  endif
enddo

! Create the bounding polygon from the input data.

if (npolygon == 0 .and. .not. rectbnd) then
  write(*,*) '***'
  write(*,*) '*** Error: Geometric boundaries for optimisation have not been specified.'
  write(*,*) '***'
  stop
else if (rectbnd .and. npolygon == 0) then
  ! Convert definition of the bounding rectangle to a 4-sided polygon.
  npolygon=4
  bndpolygon(1)=cmplx(xbndlft,ybndtop)
  bndpolygon(2)=cmplx(xbndlft,ybndbot)
  bndpolygon(3)=cmplx(xbndrgt,ybndbot)
  bndpolygon(4)=cmplx(xbndrgt,ybndtop)
  write(*,*) '*** Bounding rectangle has been converted to 4-sided polygon.'
else if (rectbnd .and. npolygon > 0) then
  write(*,*) '***'
  write(*,*) '*** Error: Both rectangular and polygonal boundaries have been specified.'
  write(*,*) '***'
  stop
else if (npolygon < 3) then
  write(*,*) '***'
  write(*,*) '*** Error: Bounding polygon must have three or more sides.'
  write(*,*) '***'
  stop
endif

write(*,*) '*** Bounding polygon with',npolygon,'sides defined.'

! Convert the definition of the bounding polygon to separate x- and y-ordinates
! for use in the code that determines whether a point lies inside a polygon.

do i=1,npolygon
  xbndpolygon(i)=real(bndpolygon(i))
  ybndpolygon(i)=imag(bndpolygon(i))
enddo

! Ensure that if the constant mesh spacing option is used, then
! it is not possible to fix the nodal movement to be in only the
! x-direction or only in the y-direction. Unless they are the
! first or the last nodes in the list, the nodes must be either
! entirely free or entirely fixed. The first or last nodes are
! exempt from this requirement because it is sometimes desirable
! to enforce symmetry conditions on the model.

if (constspacing) then
  invalidoption=.false.
  do i=2,nbndnodes-1
    if (xfixnodes(i).and..not.yfixnodes(i)) invalidoption=.true.
    if (yfixnodes(i).and..not.xfixnodes(i)) invalidoption=.true.
  enddo
  if (invalidoption) then
    write(*,*) '***'
    write(*,*) '*** Error: When the constant mesh spacing option is used, nodes'
    write(*,*) '***        other than the first and last nodes in the list of'
    write(*,*) '***        nodes must be either entirely free to move or be '
    write(*,*) '***        entirely fixed (i.e. fixed in both the x-direction'
    write(*,*) '***        and the y-direction).'
    write(*,*) '***'
    stop
  endif
```

```
endif

! Determine parameters of rework circle if it was defined.

if (reworkcircle(1) > 0.0d0) then
  rwkclrad=reworkcircle(1)
  rwkclxc=reworkcircle(2)
  rwkclyc=reworkcircle(3)
else
  rwkclrad=0.0d0
  rwkclxc=0.0d0
  rwkclyc=0.0d0
endif

! Determine the mirroring options at the node locations.

do i=1,nbndnodes
  xmirnodes(i)=.false.
  ymirnodes(i)=.false.
  write(mircode,'(i8)') mirnodes(i)
  if (index(mircode,'1') > 0) xmirnodes(i)=.true.
  if (index(mircode,'2') > 0) ymirnodes(i)=.true.
enddo

! Adjust nodal smoothing option.

if (.not.smoothxy) then
  do i=1,nbndnodes
    smthnodes(i)=0
  enddo
endif

! Check to see if the PAFEC datafile exists.

pafecfile=pafecname(1:len_trim(pafecname))//'.DAT'

if (fexist(pafecfile)) then
  resultsfile=resultsname(1:len_trim(resultsname))//'.DAT'
  joboptoutfile=resultsname(1:len_trim(resultsname))//'OPT.OUT'
  joboptsmyfile=resultsname(1:len_trim(resultsname))//'OPT.SMY'
  joboptpfcfile=resultsname(1:len_trim(resultsname))//'OPT.PFC'
  joboptscnfile=resultsname(1:len_trim(resultsname))//'OPT.SCN'
  pupoutfile=resultsname(1:len_trim(resultsname))//'.PUPPIES'
  pupjnlfile=resultsname(1:len_trim(resultsname))//'.JNL'
  bndnodesfile=resultsname(1:len_trim(resultsname))//'.NODES'
  tempfile=resultsname(1:len_trim(resultsname))//'.TEMP'
  if (.not.restart) call fcopy(pafecfile,resultsfile)
else
  write(*,*) '***'
  write(*,*) '*** Could not find '//pafecfile(1:len_trim(pafecfile))//'.'
  write(*,*) '***'
  stop
endif

! Determine whether (x,y) or (x,y,z) coordinates are supplied in
! the PAFEC data file.

open(lui,file=pafecfile,status='old')
call findstr(lui,aline,'NODE.NUMBER',ipos)
if (ipos > 0) then
  errorinxyz=.false.
  is2dgeom=.true.
  if (index(aline,'Z') > 0) then
    if (index(aline,'X') > 0 .and. index(aline,'Y') > 0) then
      if (index(aline,'Y') > index(aline,'X') .and.  &
          index(aline,'Z') > index(aline,'Y')) then
        is2dgeom=.false.
      else
        errorinxyz=.true.
      endif
    endif
  else if (index(aline,'X') > 0 .and. index(aline,'Y') > 0) then
    if (index(aline,'Y') > index(aline,'X')) then
      is2dgeom=.true.
    else
      errorinxyz=.true.
    endif
```

```
  endif
  if (errorinxyz) then
    write(*,*) '***'
    write(*,*) '*** Coordinates in NODE module must be ordered as (X,Y) or (X,Y,Z).'
    write(*,*) '***'
    stop
  endif
  if (is2dgeom) then
    write(*,*) '*** Detected 2D geometry: (X,Y) coordinates.'
  else
    write(*,*) '*** Detected 3D geometry: (X,Y,Z) coordinates.'
  endif
else
  write(*,*) '***'
  write(*,*) '*** Unable to find NODE.NUMBER column heading in PAFEC data file.'
  write(*,*) '*** Cannot determine whether 2D=(x,y) or 3D=(x,y,z) coordinates'
  write(*,*) '*** are being specified.'
  write(*,*) '***'
  stop
endif
close(lui)

! Check for user-defined equations for transforming (x,y,z) coordinates.

if (len_trim(xeqn) == 0 .and. &
    len_trim(yeqn) == 0 .and. &
    len_trim(zeqn) == 0       ) then
  usexyzeqn=.false.
else
  ! Initialise the function parser that will be used to apply the
  ! user-defined equations to the (x,y,z) coordinates.
  call initf(3)
  usexyzeqn=.true.
  if (len_trim(xeqn) > 0) then
    xyzeqn(1)=xeqn
  else
    xyzeqn(1)='xn'
  endif
  if (len_trim(yeqn) > 0) then
    xyzeqn(2)=yeqn
  else
    xyzeqn(2)='yn'
  endif
  if (len_trim(zeqn) > 0) then
    xyzeqn(3)=zeqn
  else
    xyzeqn(3)='zn'
  endif
  xyzeqnvar(1)='xn'
  xyzeqnvar(2)='yn'
  xyzeqnvar(3)='zn'
  do i=1,3
    call parsef(i,xyzeqn(i),xyzeqnvar)
  enddo
endif

call wrtjobinfo(6,1,r,s,c,nogrowth,smoothdelta,smoothxy,maxsmooth,optth,optwt,       &
                nodeth,npolygon,bndpolygon,                                          &
                rwkclrad,rwkclxc,rwkclyc,itmax,nbndnodes,nactnodes,nskiplowabs,      &
                pafecfile,resultsfile,joboptoutfile,joboptsmyfile,joboptpfcfile,     &
                chkdivergence,chkconvergence,datestamp,setmidsidenodes,              &
                title1,title2,title3,puppiesfile,minradcurvlim,programid,restorebnd, &
                constspacing,findcorners,nbnd,useglobalstress,useaveragedstress,     &
                usevonmisesstress,puppiesversion,closedbnd,multiplepeaks,stol,       &
                xeqn,yeqn,zeqn,optmodifyxyz,nuserparam,userparam,usemaxpospeaksth,   &
                useabsvaluesdr,bndlenfac)

! Perform the iterations.

if (echotoscn) then
  scn=' '
else
  scn=' > '//joboptscnfile(1:len_trim(joboptscnfile))
endif

do itnum=1,itmax
```

```
write(*,*)

call fclear(joboptscnfile(1:len_trim(joboptscnfile)))

! Run the PAFEC model.

call syscmd('pafdel '//resultsname(1:len_trim(resultsname))//scn)
call fclear(joboptscnfile(1:len_trim(joboptscnfile)))
call syscmd('rmlock -y'//scn)
call fclear(joboptscnfile(1:len_trim(joboptscnfile)))
call syscmd('pafrun '//resultsname(1:len_trim(resultsname))//scn)
call fclear(joboptscnfile(1:len_trim(joboptscnfile)))

! Check whether the PAFEC Phase 9 output file exists to see if
! the PAFEC run was successfully completed.

results09file=resultsname(1:len_trim(resultsname))//'.O09'

if (.not. fexist(results09file)) then
  runfailedmsg='*** PAFEC run failed to complete.'
  call appendmsgtofile(luo,joboptoutfile,1,runfailedmsg)
  call appendmsgtofile(luo,joboptsmyfile,1,runfailedmsg)
  write(*,*)
  write(*,*) runfailedmsg(1:len_trim(runfailedmsg))
  write(*,*)
  stop
endif

! Use PUPPIES to get the required stress data.

skipstressextraction=.false.

call getstresspup(lup,puppiesfile,pupjnlfile,resultsname,bndnodesfile,          &
                  pupoutfile,joboptscnfile,puppiesversion,useglobalstress,      &
                  useaveragedstress,scn,stopafterpuppies,skipstressextraction,  &
                  itnum,maxit,maxnode,maxbnd,nbnd,nbndnodes,bndnodes,           &
                  np,xp,yp,zp,s1,s2,s3,svm)

! Transfer some results from the data for the previous
! iteration to the current iteration.

if (itnum == 1) then
  do j=1,nbnd
    do i=1,nbndnodes
      xr(i,j,itnum)=0.0d0
      yr(i,j,itnum)=0.0d0
      zr(i,j,itnum)=0.0d0
    enddo
  enddo
else
  do j=1,nbnd
    do i=1,nbndnodes
      xr(i,j,itnum)=xn(i,j,itnum-1)
      yr(i,j,itnum)=yn(i,j,itnum-1)
      zr(i,j,itnum)=zn(i,j,itnum-1)
    enddo
  enddo
endif

! Store the stresses around the boundary in array bndstr.

if (nbnd == 1) then
  do i=1,nbndnodes
    bigs1s2s3(i,1,itnum)=biggestof3(s1(i,1,itnum),s2(i,1,itnum),s3(i,1,itnum))
  enddo
  if (usevonmisesstress) then
    do i=1,nbndnodes
      bndstr(i,itnum)=svm(i,1,itnum)
    enddo
  else
    do i=1,nbndnodes
      bndstr(i,itnum)=bigs1s2s3(i,1,itnum)
    enddo
  endif
else
  do i=1,nbndnodes
    do j=1,nbnd
```

```
        bigs1s2s3(i,j,itnum)=biggestof3(s1(i,j,itnum),s2(i,j,itnum),s3(i,j,itnum))
      enddo
    enddo
    if (usevonmisesstress) then
      ! Find the maximum von Mises stress through the
      ! thickness at each location along the boundary.
      do i=1,nbndnodes
        maxsvm=svm(i,1,itnum)
        do j=2,nbnd
          maxsvm=max(svm(i,j,itnum),maxsvm)
        enddo
        bndstr(i,itnum)=maxsvm
      enddo
    else
      ! Find the maximum principal stress occuring over the multiple boundaries.
      do i=1,nbndnodes
        maxonbndj=bigs1s2s3(i,1,itnum)
        do j=2,nbnd
          maxonbndj=biggestof2(maxonbndj,bigs1s2s3(i,j,itnum))
        enddo
        bndstr(i,itnum)=maxonbndj
      enddo
    endif
  endif

  ! Determine info related to the boundary stresses:
  ! average, maximum, minimum, range and delta.

  init=.true.
  iact=0
  bndstrtot=0.0d0
  do i=1,nbndnodes
    if (actnodes(i)) then
      if (init) then
        maxabsbndstr(itnum)=abs(bndstr(i,itnum))
        minabsbndstr(itnum)=abs(bndstr(i,itnum))
        maxbndstr(itnum)=bndstr(i,itnum)
        minbndstr(itnum)=bndstr(i,itnum)
        init=.false.
      endif
      maxabsbndstr(itnum)=max(maxabsbndstr(itnum),abs(bndstr(i,itnum)))
      minabsbndstr(itnum)=min(minabsbndstr(itnum),abs(bndstr(i,itnum)))
      maxbndstr(itnum)=max(maxbndstr(itnum),bndstr(i,itnum))
      minbndstr(itnum)=min(minbndstr(itnum),bndstr(i,itnum))
      bndstrtot=bndstrtot+bndstr(i,itnum)
      iact=iact+1
      actbndstr(iact)=bndstr(i,itnum)
    endif
  enddo
  pcinit(itnum)=maxabsbndstr(itnum)/maxabsbndstr(1)*100.0d0
  avgbndstr(itnum)=bndstrtot/nactnodes
  rngbndstr(itnum)=maxbndstr(itnum)-minbndstr(itnum)
  rngabsbndstr(itnum)=maxabsbndstr(itnum)-minabsbndstr(itnum)
  if (itnum == 1) then
    dltbndstr(itnum)=0.0d0
    dltrngbndstr(itnum)=0.0d0
  else
    dltbndstr(itnum)=maxabsbndstr(itnum)-maxabsbndstr(itnum-1)
    dltrngbndstr(itnum)=rngbndstr(itnum)-rngbndstr(itnum-1)
  endif

  ! Calculate average of absolute values of the boundary stress.
  ! Skip values with a low absolute value if required.

  call dcopyvec(actbndstr,sortabsbndstr,1,nactnodes)
  call dabsvec(sortabsbndstr,nactnodes)
  call dabsord(sortabsbndstr,nactnodes)
  avgabsbndstr(itnum)=davgvec(sortabsbndstr(1+nskiplowabs),nactnodes-nskiplowabs)

  ! Determine locations of any zero crossings in the stored boundary stresses
  ! and the values of the peak and average stresses lying between any of these
  ! zero crossings. If no zero crossings are present, then the peak and average
  ! stresses occurring on the interval are determined.

  if (closedbnd) then
    call getzerocrossclosed(nbndnodes,bndstr(1,itnum),nzc,izcbeg,izcend,nzones(itnum),izone)
  else
```

```
  call getzerocross(nbndnodes,bndstr(1,itnum),nzc,izcbeg,izcend,nzones(itnum),izone)
endif
call getpkavgbwzone(nbndnodes,bndstr(1,itnum),nzones(itnum),izone,zonepeak(1,itnum), &
                    zoneavg(1,itnum),izonepeak)
call getstatszone(nbndnodes,bndstr(1,itnum),nzones(itnum),izone,stol,zonepeak(1,itnum), &
                  zonemin(1,itnum),zoneavg(1,itnum),zonesd(1,itnum),zonerange(1,itnum), &
                  izonepeak)

! Set up the stress threshold that will be used.

if (nodeth > 0) then
   sth(1,itnum)=bndstr(inodeth,itnum)
else if (optth == 0) then
  if (multiplepeaks) then
    if (usemaxpospeaksth) then
      maxpospeak=dmaxvec(zonepeak(1,itnum),nzones(itnum))
      do i=1,nzones(itnum)
        if (zonepeak(i,itnum) > 0.0d0) then
          sth(i,itnum)=maxpospeak
        else
          sth(i,itnum)=zonepeak(i,itnum)
        endif
      enddo
    else
      do i=1,nzones(itnum)
        sth(i,itnum)=zonepeak(i,itnum)
      enddo
    endif
  else
    sth(1,itnum)=maxbndstr(itnum)
  endif
else if (optth == 1) then
  if (multiplepeaks) then
    do i=1,nzones(itnum)
      sth(i,itnum)=zoneavg(i,itnum)
    enddo
  else
    sth(1,itnum)=avgabsbndstr(itnum)
  endif
else if (optth == 2) then
  if (multiplepeaks) then
    write(*,*) '*** optth=2 and multiplepeaks capability not implemented.'
    stop
  else
    sth(1,itnum)=(maxabsbndstr(itnum)+minabsbndstr(itnum))/2.0d0
  endif
else if (optth == 4) then
  if (multiplepeaks) then
    write(*,*) '*** optth=4 and multiplepeaks capability not implemented.'
    stop
  else
    sth(1,itnum)=(maxabsbndstr(itnum)+avgabsbndstr(itnum))/2.0d0
  endif
else
  write(*,*) '***'
  write(*,*) '*** Undefined option for stress threshold was used.'
  write(*,*) '*** optth=',optth
  write(*,*) '***'
  stop
endif

! For the boundary shape as read in from the last analysis results, compute the
! local radius of curvature and determine if the local shape is concave.

if (closedbnd) then
  call radcurvcircclosed(nbndnodes,xp(1,1,itnum),yp(1,1,itnum), &
                         radcurv(1,itnum),10000.0d0,clockwise,  &
                         concave(1,itnum),elementsonright)
else
  call radcurvcirc(nbndnodes,xp(1,1,itnum),yp(1,1,itnum),          &
                   xmirnodes,ymirnodes,radcurv(1,itnum),10000.0d0, &
                   clockwise,concave(1,itnum),elementsonright)
endif

! Compute statistics related to radius of curvature at each local nodal point.

call dvecmaxact(nbndnodes,radcurv(1,itnum),actnodes,maxradcurv(itnum))
```

```
call dvecminact(nbndnodes,radcurv(1,itnum),actnodes,minradcurv(itnum))
call dvecavgact(nbndnodes,radcurv(1,itnum),actnodes,avgradcurv(itnum))

! Determine non-dimensional arc length spacing for the nodes on the moving boundary.

if (closedbnd) then
  call polygarclenc(nbndnodes,xp(1,1,itnum),yp(1,1,itnum),scurbnd,tcurbnd,scurbndtotal)
else
  call polygarclen(nbndnodes,xp(1,1,itnum),yp(1,1,itnum),scurbnd,tcurbnd)
  scurbndtotal = scurbnd(nbndnodes)
endif

! Determine length of the moving boundary within each zone of uniform stress.
! Note that a closed boundary needs special treatment at the end to get the
! summation correct.

do i=1,nzones(itnum)
  zonet(i,itnum)=0.0d0
enddo
do i=1,nbndnodes-1
  ! Check that both nodes are in the same stress zone.
  if (izone(i) == izone(i+1)) then
    snodei  =bndstr(i  ,itnum)
    snodeip1=bndstr(i+1,itnum)
    if (within(snodei  ,zonepeak(izone(i  ),itnum),stol,'[]') .and.  &
        within(snodeip1,zonepeak(izone(i+1),itnum),stol,'[]')      ) then
      zonet(izone(i),itnum)=zonet(izone(i),itnum)+abs(tcurbnd(i)-tcurbnd(i+1))
    endif
  endif
enddo
if (closedbnd) then
  if (izone(nbndnodes) == izone(1)) then
    snodei  =bndstr(nbndnodes,itnum)
    snodeip1=bndstr(1        ,itnum)
    if (within(snodei  ,zonepeak(izone(nbndnodes),itnum),stol,'[]') .and.  &
        within(snodeip1,zonepeak(izone(1        ),itnum),stol,'[]')      ) then
      zonet(izone(nbndnodes),itnum)=zonet(izone(nbndnodes),itnum) &
                                    +abs(tcurbnd(nbndnodes)-1.0d0)
    endif
  endif
endif

zonetotlen(itnum)=dsumvec(zonet(1,itnum),nzones(itnum))

nzonesmax=imaxvec(nzones,itnum)

! Display parameters on the screen at each iteration.

elapsedtime=(time()-time0)/3600.0d0 ! in hours

write(*,1) '*** Iteration number                = ',itnum
write(*,2) '*** Percentage of initial stress    = ',pcinit(itnum),' %'
write(*,2) '*** Stress threshold                = ',sth(1,itnum)
write(*,2) '*** Max stress                      = ',maxbndstr(itnum)
write(*,2) '*** Min stress                      = ',minbndstr(itnum)
write(*,2) '*** Range stress                    = ',rngbndstr(itnum)
write(*,2) '*** Avg absolute stress             = ',avgabsbndstr(itnum)
write(*,2) '*** Delta stress                    = ',dltbndstr(itnum)
write(*,1) '*** Number of stress zones          = ',nzones(itnum)
write(*,3) '*** Peak stress in each zone        = ',(zonepeak(i,itnum),i=1,nzones(itnum))
if (multiplepeaks .and. itnum == 1) then
  write(*,3) '*** Unmodified stress thresholds    = ',(sth(i,itnum),i=1,nzones(itnum))
  do i=1,nzones(itnum)
    sth(i,itnum)=strthreshfac(i)*sth(i,itnum)
  enddo
  write(*,3) '*** Modified stress thresholds      = ',(sth(i,itnum),i=1,nzones(itnum))
endif
write(*,3) '*** Minimum stress in zone          = ',(zonemin(i,itnum),i=1,nzones(itnum))
write(*,3) '*** Average stress in zone          = ',(zoneavg(i,itnum),i=1,nzones(itnum))
write(*,3) '*** Range stress in zone            = ',(zonerange(i,itnum),i=1,nzones(itnum))
write(*,3) '*** Std dev stress in zone          = ',(zonesd(i,itnum),i=1,nzones(itnum))
write(*,5) '*** Uniform zone length             = ',(zonet(i,itnum),i=1,nzones(itnum))
write(*,2) '*** Total uniform length            = ',zonetotlen(itnum)
write(*,2) '*** Max radius of curvature         = ',maxradcurv(itnum)
write(*,2) '*** Min radius of curvature         = ',minradcurv(itnum)
write(*,2) '*** Elapsed time                    = ',elapsedtime,' hours'
```

```
1       format(1x,a,i13)
2       format(1x,a,f13.6,a)
3       format(1x,a,<nzones(itnum)>f10.3)
4       format(1x,a,a13)
5       format(1x,a,<nzones(itnum)>f10.4)

        ! Check whether all the nodes that were read in from the last PAFEC run are
        ! within the bounding polygon.

        call checkxypolygon(allpointsinside,xpypinside,nbndnodes,np(1,1,itnum), &
                            xp(1,1,itnum),yp(1,1,itnum),actnodes,               &
                            npolygon,xbndpolygon,ybndpolygon,.true.)

        if (.not.allpointsinside .and. itnum == 1) stop

        ! Now calculate dr, the distance to be moved.
        ! dr is positive if material is to be added.
        ! dr is negative if material is to be removed.

        maxdist(itnum)=0.0d0
        do i=1,nbndnodes
          if (multiplepeaks) then
            if (useabsvaluesdr) then
              deltabndstr(i,itnum)=abs(bndstr(i,itnum))-abs(sth(izone(i),itnum))
            else
              deltabndstr(i,itnum)=bndstr(i,itnum)-sth(izone(i),itnum)
            endif
          else
            if (useabsvaluesdr) then
              deltabndstr(i,itnum)=abs(bndstr(i,itnum))-abs(sth(1,itnum))
            else
              deltabndstr(i,itnum)=bndstr(i,itnum)-sth(1,itnum)
            endif
          endif
          if (actnodes(i)) then
            if (optwt == 1) then
              sr=abs(bndstr(i,itnum))/abs(maxbndstr(itnum))
              wt=sin(sr*pion2)**2
            else
              wt=1.0
            endif
            if (multiplepeaks) then
              if (useabsvaluesdr) then
                dr(i,itnum)=(deltabndstr(i,itnum)/abs(sth(izone(i),itnum)))*r*s*wt
              else
                dr(i,itnum)=(deltabndstr(i,itnum)/sth(izone(i),itnum))*r*s*wt
              endif
            else
              if (useabsvaluesdr) then
                dr(i,itnum)=(deltabndstr(i,itnum)/abs(sth(1,itnum)))*r*s*wt
              else
                dr(i,itnum)=(deltabndstr(i,itnum)/sth(1,itnum))*r*s*wt
              endif
            endif
            if (nogrowth.and.dr(i,itnum) > 0.0d0) then
              dr(i,itnum)=0.0d0
            endif
            maxdist(itnum)=max(maxdist(itnum),abs(dr(i,itnum)))
          else
            dr(i,itnum)=0.0d0
          endif
        enddo

        write(*,2) '*** Max unsmoothed nodal movement    = ',maxdist(itnum)

        ! If required, smooth the delta values to be applied to the
        ! nodal coordinates (using simple 3-point smoothing).

        if (smoothdelta .and. itnum <= maxsmooth) then
          if (closedbnd) then
            drs(1)=(dr(nbndnodes,itnum)+dr(1,itnum)+dr(2,itnum))/3
            drs(nbndnodes)=(dr(nbndnodes-1,itnum)+dr(nbndnodes,itnum)+dr(1,itnum))/3
          else
            drs(1)=dr(1,itnum)
            drs(nbndnodes)=dr(nbndnodes,itnum)
          endif
          do i=2,nbndnodes-1
```

```
    drs(i)=(dr(i-1,itnum)+dr(i,itnum)+dr(i+1,itnum))/3
  enddo
  do i=1,nbndnodes
    dr(i,itnum)=drs(i)
  enddo
  maxdist(itnum)=0.0d0
  do i=1,nbndnodes
    maxdist(itnum)=max(maxdist(itnum),abs(dr(i,itnum)))
  enddo
  write(*,2) '*** Max smoothed nodal movement      = ',maxdist(itnum)
endif

! Change the coordinates by moving each node in the direction of the local outward
! normal to the surface. However, if the x- or y-ordinate is fixed, handle this
! in a special way.

do i=1,nbndnodes
  ! Compute angle of slope of the curve at the local point.
  if (i == 1) then
    if (closedbnd) then
      angle=atan2(yp(i+1,1,itnum)-yp(nbndnodes,1,itnum), &
                  xp(i+1,1,itnum)-xp(nbndnodes,1,itnum))
    else
      if (xmirnodes(i)) then
        angle=atan2(0.0d0,xp(i+1,1,itnum)-(-xp(i+1,1,itnum)))
      else if (ymirnodes(i)) then
        angle=atan2(yp(i+1,1,itnum)-(-yp(i+1,1,itnum)),0.0d0)
      else
        angle=atan2(yp(i+1,1,itnum)-yp(i,1,itnum),xp(i+1,1,itnum)-xp(i,1,itnum))
      endif
    endif
  else if (i == nbndnodes) then
    if (closedbnd) then
      angle=atan2(yp(1,1,itnum)-yp(i-1,1,itnum),xp(1,1,itnum)-xp(i-1,1,itnum))
    else
      if (xmirnodes(i)) then
        angle=atan2(0.0d0,-xp(i-1,1,itnum)-xp(i-1,1,itnum))
      else if (ymirnodes(i)) then
        angle=atan2(-yp(i-1,1,itnum)-yp(i-1,1,itnum),0.0d0)
      else
        angle=atan2(yp(i,1,itnum)-yp(i-1,1,itnum),xp(i,1,itnum)-xp(i-1,1,itnum))
      endif
    endif
  else
    ! This simple formula for calculating the slope of the curve is equivalent to
    ! what is obtained by using a second-order Lagrange polynomial approximation.
    angle=atan2(yp(i+1,1,itnum)-yp(i-1,1,itnum),xp(i+1,1,itnum)-xp(i-1,1,itnum))
  endif
  ! Compute angle of the outward normal.
  theta(i,itnum)=angle+pion2
  if (theta(i,itnum) > pi) then
    theta(i,itnum)=theta(i,itnum)-twopi
  endif
  ! Compute new nodal coordinates.
  if (actnodes(i)) then
    if (.not. xpypinside(i)) then
      xu(i)=xp(i,1,itnum)
      yu(i)=yp(i,1,itnum)
    else if (xfixnodes(i).and.yfixnodes(i)) then
      xu(i)=xp(i,1,1)
      yu(i)=yp(i,1,1)
    else if (xfixnodes(i)) then
      xu(i)=xp(i,1,itnum)
      yu(i)=yp(i,1,itnum)+dr(i,itnum)*dsign(1.0d0,sin(theta(i,itnum)))
    else if (yfixnodes(i)) then
      xu(i)=xp(i,1,itnum)+dr(i,itnum)*dsign(1.0d0,cos(theta(i,itnum)))
      yu(i)=yp(i,1,itnum)
    else
      xu(i)=xp(i,1,itnum)+dr(i,itnum)*cos(theta(i,itnum))
      yu(i)=yp(i,1,itnum)+dr(i,itnum)*sin(theta(i,itnum))
    endif
  else
    xu(i)=xp(i,1,1)
    yu(i)=yp(i,1,1)
  endif
enddo
```

```
! Check to see if rework circle limits are violated by any nodes.
! If they are, set the offending nodes to lie on the boundary
! of the rework circle.

if (rwkclrad > 0.0d0) then
  do i=1,nbndnodes
    if (actnodes(i) .and. xpypinside(i)) then
      radius=sqrt((xu(i)-rwkclxc)**2+(yu(i)-rwkclyc)**2)
      issamepoint=(xu(i) == xp(i,1,itnum)) .and. (yu(i) == yp(i,1,itnum))
      if (radius < rwkclrad .and. .not. issamepoint) then
        ! Determine intersection points for the line that joins
        ! the old and the new point and the rework circle.
        call intersectlinesphere(xu(i),yu(i),0.0d0,xp(i,1,itnum),yp(i,1,itnum),0.0d0, &
                                 rwkclrad,rwkclxc,rwkclyc,0.0d0,                      &
                                 ni,xint1,yint1,zint1,xint2,yint2,zint2)
        if (ni == 2) then
          ! Determine which intersection point falls on the
          ! line segment and set the nodal coordinates to
          ! coincide with this point.
          dst1=sqrt((xp(i,1,itnum)-xint1)**2+(yp(i,1,itnum)-yint1)**2)
          dst2=sqrt((xp(i,1,itnum)-xint2)**2+(yp(i,1,itnum)-yint2)**2)
          if (dst1 < dst2) then
            xu(i)=xint1
            yu(i)=yint1
          else
            xu(i)=xint2
            yu(i)=yint2
          endif
        else if (ni == 1) then
          xu(i)=xint1
          yu(i)=yint1
        else if (ni == 0) then
          write(*,*) '***'
          write(*,*) '*** Error: No intersection with rework circle.'
          write(*,*) '***'
          stop
        endif
      endif
    endif
  enddo
endif

! Use a three-point moving average to smooth the boundary coordinates
! to try and prevent a jagged boundary from occurring.

if (smoothxy .and. itnum <= maxsmooth) then
  if (closedbnd .and. smthnodes(1) == 1) then
    xs(i)=(xu(nbndnodes)+xu(1)+xu(2))/3
    ys(i)=(yu(nbndnodes)+yu(1)+yu(2))/3
  else
    xs(1)=xu(1)
    ys(1)=yu(1)
  endif
  do i=2,nbndnodes-1
    if (smthnodes(i) == 1) then
      xs(i)=(xu(i-1)+xu(i)+xu(i+1))/3
      ys(i)=(yu(i-1)+yu(i)+yu(i+1))/3
    else
      xs(i)=xu(i)
      ys(i)=yu(i)
    endif
  enddo
  if (closedbnd .and. smthnodes(nbndnodes) == 1) then
    xs(i)=(xu(nbndnodes-1)+xu(nbndnodes)+xu(1))/3
    ys(i)=(yu(nbndnodes-1)+yu(nbndnodes)+yu(1))/3
  else
    xs(nbndnodes)=xu(nbndnodes)
    ys(nbndnodes)=yu(nbndnodes)
  endif
  do i=1,nbndnodes
    xn(i,1,itnum)=xs(i)
    yn(i,1,itnum)=ys(i)
  enddo
  ! Reset x- and/or y-ordinates if these were fixed, or if they
  ! lie outside the bounding polygon.
  do i=1,nbndnodes
    if (xfixnodes(i)) xn(i,1,itnum)=xp(i,1,1)
```

```
    if (yfixnodes(i)) yn(i,1,itnum)=yp(i,1,1)
    if (xpypinside(i)) then
      xn(i,1,itnum)=xp(i,1,itnum)
      yn(i,1,itnum)=yp(i,1,itnum)
    endif
  enddo
else
  do i=1,nbndnodes
    xn(i,1,itnum)=xu(i)
    yn(i,1,itnum)=yu(i)
  enddo
endif

! Enforce the polygonal constraint boundary.

call makexypolygon(nbndnodes,xn(1,1,itnum),yn(1,1,itnum),xpypinside,             &
                   xp(1,1,itnum),yp(1,1,itnum),npolygon,xbndpolygon,ybndpolygon, &
                   actnodes,xfixnodes,yfixnodes,wasconstrained)

if (wasconstrained) then
  write(*,4) '*** Polygon constraints enforced     = ','Yes'
else
  write(*,4) '*** Polygon constraints enforced     = ','No'
endif

! Adjust the local nodal positions if the minimum radius of curvature
! limit has been violated at any nodes along the new boundary.

nrcpass=0
maxrcpass=48
if (minradcurvlim > 0.0d0 .and. restorebnd) then
  minrc=0.0d0
  do while (minrc < minradcurvlim .and. nrcpass <= maxrcpass)
    nrcpass=nrcpass+1
    ! Compute the radius of curvature at the nodes for the new boundary.
    if (closedbnd) then
      call radcurvcircclosed(nbndnodes,xn(1,1,itnum),yn(1,1,itnum),     &
                             radcurvnew,10000.0d0,clockwise,concavenew, &
                             elementsonright)
    else
      call radcurvcirc(nbndnodes,xn(1,1,itnum),yn(1,1,itnum),    &
                       xmirnodes,ymirnodes,radcurvnew,10000.0d0, &
                       clockwise,concavenew,elementsonright)
    endif
    ! Determine minimum radius of curvature for concave sections
    ! of the new boundary that are active.
    call dvecminactconc(nbndnodes,radcurvnew,actnodes,concavenew,minrc)
    ! Adjust locations of those nodes where the radius of curvature is less than
    ! the specified minimum permissible radius of curvature.
    if (minrc < minradcurvlim) then
      do i=1,nbndnodes
        if (actnodes(i) .and. concavenew(i) == 1 .and.           &
            radcurvnew(i) <  minradcurvlim .and. xpypinside(i) ) then
          if (xfixnodes(i).and.yfixnodes(i)) then
            xn(i,1,itnum)=xp(i,1,1)
            yn(i,1,itnum)=yp(i,1,1)
          else if (xfixnodes(i)) then
            xn(i,1,itnum)=xp(i,1,itnum)
            yn(i,1,itnum)=(yp(i,1,itnum)+yn(i,1,itnum))/2.0d0
            if (nrcpass == maxrcpass) then
              yn(i,1,itnum)=yp(i,1,itnum)
            endif
          else if (yfixnodes(i)) then
            xn(i,1,itnum)=(xp(i,1,itnum)+xn(i,1,itnum))/2.0d0
            yn(i,1,itnum)=yp(i,1,itnum)
            if (nrcpass == maxrcpass) then
              xn(i,1,itnum)=xp(i,1,itnum)
            endif
          else
            xn(i,1,itnum)=(xp(i,1,itnum)+xn(i,1,itnum))/2.0d0
            yn(i,1,itnum)=(yp(i,1,itnum)+yn(i,1,itnum))/2.0d0
            if (nrcpass == maxrcpass) then
              xn(i,1,itnum)=xp(i,1,itnum)
              yn(i,1,itnum)=yp(i,1,itnum)
            endif
          endif
        endif
```

```
      enddo
    endif
  enddo
  write(*,1) '*** Number passes radius curvature   = ',nrcpass
endif

! Adjust the nodal positions so that the original element spacing is maintained.

if (constspacing) then
  if (closedbnd) then
    ! Determine the initial spacing of the nodes along the moving boundary.
    if (itnum == 1) then
      call polygarclenc(nbndnodes,xp(1,1,itnum),yp(1,1,itnum),sbnd,tbnd,sbndtotal)
    endif
    ! Redistribute the iterated nodes along the moving boundary so that
    ! the distribution is the same as that in the initial model.
    sigmakurvp=0.0d0
    call kurvp1(nbndnodes,xn(1,1,itnum),yn(1,1,itnum),xpkurvp,ypkurvp, &
                tempkurvp,skurvp,sigmakurvp,ierrkurvp)
    if (ierrkurvp /= 0) then
      write(*,*)
      write(*,*) '*** Error in kurvp1 while adjusting moving boundary mesh spacing.'
      write(*,*) '*** Error code =',ierrkurv,'(adjacent coordinate pairs coincide).'
      write(*,*)
      write(*,'(1x,a6,2a15)') 'Node','X','Y'
      do i=1,nbndnodes
        write(*,'(1x,i6,1p2e15.6)') bndnodes(i,1),xn(i,1,itnum),yn(i,1,itnum)
      enddo
      write(*,*)
      stop
    endif
    do i=1,nbndnodes
      call kurvp2(tbnd(i),xnew(i),ynew(i),nbndnodes,xn(1,1,itnum),yn(1,1,itnum), &
                  xpkurvp,ypkurvp,skurvp,sigmakurvp)
    enddo
  else
    ! Determine the initial spacing of the nodes along the moving boundary.
    if (itnum == 1) then
      call polygarclen(nbndnodes,xp(1,1,itnum),yp(1,1,itnum),sbnd,tbnd)
    endif
    ! Redistribute the iterated nodes along the moving boundary so that
    ! the distribution is the same as that in the initial model.
    sigmakurv=0.0d0
    isfkurv=3
    call kurv1(nbndnodes,xn(1,1,itnum),yn(1,1,itnum),slope1kurv,slopenkurv, &
               isfkurv,xpkurv,ypkurv,tempkurv,skurv,sigmakurv,ierrkurv)
    if (ierrkurv /= 0) then
      write(*,*) '***'
      write(*,*) '*** Error in kurv1 while adjusting moving boundary mesh spacing.'
      write(*,*) '*** Error code =',ierrkurv,'(adjacent coordinate pairs coincide).'
      write(*,*) '***'
      write(*,'(1x,a6,2a15)') 'Node','X','Y'
      do i=1,nbndnodes
        write(*,'(1x,i6,1p2e15.6)') bndnodes(i,1),xn(i,1,itnum),yn(i,1,itnum)
      enddo
      write(*,*) '***'
      stop
    endif
    do i=1,nbndnodes
      call kurv2(tbnd(i),xnew(i),ynew(i),nbndnodes,xn(1,1,itnum),yn(1,1,itnum), &
                 xpkurv,ypkurv,skurv,sigmakurv)
    enddo
  endif
  call dcopyvec(xnew,xn(1,1,itnum),1,nbndnodes)
  call dcopyvec(ynew,yn(1,1,itnum),1,nbndnodes)
  ! Reset x- and y-ordinates if these were fixed or if the node
  ! is not active during the optimisation.
  do i=1,nbndnodes
    if ((xfixnodes(i).and.yfixnodes(i)) .or. .not.actnodes(i)) then
      xn(i,1,itnum)=xp(i,1,1)
      yn(i,1,itnum)=yp(i,1,1)
    endif
  enddo
endif

! Adjust mesh spacing along edges if corners are present.
```

```
      if (.not. closedbnd .and. findcorners .and. itnum >= itcorner) then
        ! Scan the nodes and identify any corners if present.
        ncorners=0
        iscorner(1)=.false.
        iscorner(nbndnodes)=.false.
        do i=2,nbndnodes-1
          call includedangle(xn(i-1,1,itnum),yn(i-1,1,itnum),0.0d0, &
                             xn(i+0,1,itnum),yn(i+0,1,itnum),0.0d0, &
                             xn(i+1,1,itnum),yn(i+1,1,itnum),0.0d0, &
                             inclangle(i))
          if (inclangle(i) < anglecorner) then
            iscorner(i)=.true.
            ncorners=ncorners+1
            icorners(ncorners)=i
          else
            iscorner(i)=.false.
          endif
        enddo
        write(*,1) '*** Number of corners found          = ',ncorners
        ! Interpolate the nodes along each edge that terminates or starts at a corner.
        if (ncorners > 0) then
          write(*,6) &
            '*** List of nodes at corners        = ',(bndnodes(icorners(i),1),i=1,ncorners)
          write(*,7) &
            '*** Included angle at each corner   = ',(inclangle(icorners(i)),i=1,ncorners)
6         format(1x,a,7x,<ncorners>i6)
7         format(1x,a,7x,<ncorners>f6.1)
          ! Determine number of nodes along each edge.
          iedgebeg(1)=1
          do i=1,ncorners
            iedgeend(i)=icorners(i)
            iedgebeg(i+1)=icorners(i)
          enddo
          iedgeend(ncorners+1)=nbndnodes
          do i=1,ncorners+1
            nnodeedge(i)=iedgeend(i)-iedgebeg(i)+1
          enddo
          ! Redistribute the nodes along each edge.
          do i=1,ncorners+1
            ! Compute parametric spline representation of the edge.
            sigmakurv=0.0d0
            isfkurv=3
            call kurv1(nnodeedge(i),xn(iedgebeg(i),1,itnum),yn(iedgebeg(i),1,itnum), &
                       slope1kurv,slopenkurv,isfkurv,xpkurv,ypkurv,                  &
                       tempkurv,skurv,sigmakurv,ierrkurv)
            if (ierrkurv /= 0) then
              write(*,*) '***'
              write(*,*) '*** Error in kurv1 while computing mesh when corners present.'
              write(*,*) '*** Error code =',ierrkurv
              write(*,*) '***'
              stop
            endif
            ! Compute the new nodal coordinates along each edge.
            do j=1,nnodeedge(i)
              dt=1.0d0/(nnodeedge(i)-1)
              call kurv2((j-1)*dt,xnew(j),ynew(j),nnodeedge(i),           &
                         xn(iedgebeg(i),1,itnum),yn(iedgebeg(i),1,itnum), &
                         xpkurv,ypkurv,skurv,sigmakurv)
            enddo
            ! Update the vector storing the nodal coordinates with the new values.
            call dcopyvec(xnew,xn(iedgebeg(i),1,itnum),1,nnodeedge(i))
            call dcopyvec(ynew,yn(iedgebeg(i),1,itnum),1,nnodeedge(i))
          enddo
        endif
      endif

      ! Set the locations of the midside nodes of the PAFBLOCKS so that they
      ! are exactly located at the centre of the arc that defines the edge
      ! of the PAFBLOCK along the boundary that is being optimised.

      if (.not. closedbnd .and. setmidsidenodes) then
        do i=1,nbndnodes-2,2
          call settomid(xn(i+0,1,itnum),yn(i+0,1,itnum), &
                        xn(i+1,1,itnum),yn(i+1,1,itnum), &
                        xn(i+2,1,itnum),yn(i+2,1,itnum))
        enddo
      endif
```

```
! Check to see if there are any nodes that lie outside the polygon.

allpointsinside=.true.
do i=1,nbndnodes
  call dlocpt(xn(i,1,itnum),yn(i,1,itnum),xbndpolygon,ybndpolygon,npolygon,l,m)
  if (l == -1) then
    allpointsinside=.false.
    write(*,1) '*** New shape: node outside polygon  = ',np(i,1,itnum)
  endif
enddo

! Complete the (x,y,z) coordinates for boundary lines.

do j=1,nbnd
  do i=1,nbndnodes
    if (j > 1) then
      xn(i,j,itnum)=xp(i,1,itnum)
      yn(i,j,itnum)=yp(i,1,itnum)
    endif
    zn(i,j,itnum)=zp(i,j,itnum)
  enddo
enddo

! Apply the user-defined equations transforming the (x,y,z) coordinates.

if (usexyzeqn) then
  do j=1,nbnd
    do i=1,nbndnodes
      xyzeqnval(1)=xn(i,j,itnum)
      xyzeqnval(2)=yn(i,j,itnum)
      xyzeqnval(3)=zn(i,j,itnum)
      xn(i,j,itnum)=evalf(1,xyzeqnval)
      yn(i,j,itnum)=evalf(2,xyzeqnval)
      zn(i,j,itnum)=evalf(3,xyzeqnval)
    enddo
  enddo
endif

! Apply selected option transforming the (x,y,z) coordinates.

if (optmodifyxyz == 1) then
  ! Check that the expected variables have been set.
  if (nuserparam /= 3) then
    write(*,*) '*** Error: invalid number of user-defined parameters'
    write(*,*) '***        supplied for option optmodifyxyz = 1.'
    write(*,*) '***        nuserparam = ',nuserparam
    stop
  endif
  b =userparam(1)
  ts=userparam(2)
  rf=userparam(3)
  af=b-ts-rf
  do j=1,nbnd
    do i=1,nbndnodes
      if (zn(i,j,itnum) >= 0.0d0) then
        if (inrange(yn(i,j,itnum),-(b-ts),-af)) then
          zn(i,j,itnum)=rf-sqrt(rf**2-(yn(i,j,itnum)+af)**2)
        else
          zn(i,j,itnum)=0.0d0
        endif
      endif
    enddo
  enddo
endif

! Apply selected transform to the (x,y.z) coordinates.

if (optmodifyxyz == 1) then
endif

! Compute the resolved components of the displacement, as well as
! the magnitude of the distance moved.

do i=1,nbndnodes
  xd(i,itnum)=xn(i,1,itnum)-xp(i,1,itnum)
  yd(i,itnum)=yn(i,1,itnum)-yp(i,1,itnum)
```

```
  da(i,itnum)=sqrt(xd(i,itnum)**2+yd(i,itnum)**2)
enddo

! Process the supplementary nodes.

if (nsupnodes > 0) then
  ! Ensure that stress extraction is skipped so as not to overwrite the existing data.
  ! The coordinates of the master nodes are obtained even though their nodal locations
  ! are not required.
  write(*,*) '***'
  write(*,*) '*** Processing and modifying coordinates of supplementary nodes.'
  skipstressextraction=.true.
  call getstresspup(lup,puppiesfile,pupjnlfile,resultsname,bndnodesfile,          &
                    pupoutfile,joboptscnfile,puppiesversion,useglobalstress,      &
                    useaveragedstress,scn,stopafterpuppies,skipstressextraction,  &
                    itnum,maxit,maxnode,maxbnd,nsupbnd,nsupnodes,supnodes,        &
                    npsup,xpsup,ypsup,zpsup,s1,s2,s3,svm)
  ! Modify x-ordinates of selected nodes to match the new locations
  ! of each of the master node along the boundary.
  do j=2,nsupbnd
    do i=1,nsupnodes
      xpsup(i,j,itnum)=xn(ipsupnode(i),jpsupnode(i),itnum)
    enddo
  enddo
  do i=1,nsupnodes
    xpsup(i,1,itnum)=xn(ipsupnode(i),jpsupnode(i),itnum)
    ypsup(i,1,itnum)=yn(ipsupnode(i),jpsupnode(i),itnum)
    zpsup(i,1,itnum)=zn(ipsupnode(i),jpsupnode(i),itnum)
  enddo
endif

! Check for convergence based on the deltabndstr of nodes that
! are still actively moving.

if (chkconvergence) then
  if (chkxbnd .and. chkybnd) then
    solved=.true.
    do i=1,nbndnodes
      if (actnodes(i)) then
        if (xu(i) > xbndlft.and.xu(i) < xbndrgt .and. &
            yu(i) > ybndbot.and.yu(i) < ybndtop       ) then
          if (multiplepeaks) then
            if (abs(deltabndstr(i,itnum)/sth(izone(i),itnum)) > c) solved=.false.
          else
            if (abs(deltabndstr(i,itnum)/sth(1,itnum)) > c) solved=.false.
          endif
        endif
      endif
    enddo
    ! Check the last few values of the maximum boundary stress to see
    ! if they are identical (indicating a converged solution).
    if (.not.solved .and. itnum > 2 .and. .not.multiplepeaks) then
      solved=.true.
      do i=1,2
        if (maxbndstr(itnum) /= maxbndstr(itnum-i)) then
          solved=.false.
        endif
      enddo
    endif
  endif
else
  solved=.false.
endif

! Check whether the solution is diverging by looking at the last few results
! to see if they are all increasing in value instead of decreasing. Some
! oscillation can occur, so this is not always a reliable test. However, it
! works in many instances and saves on unnecessary iterations taking place.

if (chkdivergence.and.itnum > maxdivg) then
  jobdiverging=.true.
  do i=itnum,itnum-maxdivg,-1
    if (maxabsbndstr(i) <= maxabsbndstr(i-1)) then
      jobdiverging=.false.
    endif
  enddo
else
```

```
  jobdiverging=.false.
endif

! Compute some data concerning which iteration produced the best results
! on the basis of lowest peak stress and minimum stress range.

minabsbndstrit=maxabsbndstr(1)
iminabsbndstrit=1
minmaxbndstr=maxbndstr(1)
iminmaxbndstr=1
minrngbndstr=rngbndstr(1)
iminrngbndstr=1
minrngabsbndstr=rngabsbndstr(1)
iminrngabsbndstr=1
maxzonetotlen=zonetotlen(1)
imaxzonetotlen=1
do j=1,itnum
  if (maxbndstr(j) < minmaxbndstr) then
    iminmaxbndstr=j
    minmaxbndstr=maxbndstr(j)
  endif
  if (maxabsbndstr(j) < minabsbndstrit) then
    iminabsbndstrit=j
    minabsbndstrit=maxabsbndstr(j)
  endif
  if (rngbndstr(j) < minrngbndstr) then
    iminrngbndstr=j
    minrngbndstr=rngbndstr(j)
  endif
  if (rngabsbndstr(j) < minrngabsbndstr) then
    iminrngabsbndstr=j
    minrngabsbndstr=rngabsbndstr(j)
  endif
  if (zonetotlen(j) > maxzonetotlen) then
    imaxzonetotlen=j
    maxzonetotlen=zonetotlen(j)
  endif
enddo

! Adjust the x- and y-ordinates of nodes along the boundaries other than the
! first. Make them the same as those assigned to the first boundary contour.

do j=2,nbnd
  do i=1,nbndnodes
    xn(i,j,itnum)=xn(i,1,itnum)
    yn(i,j,itnum)=yn(i,1,itnum)
  enddo
enddo

! Write summary of results to the job summary file.

open (unit=luo,file=joboptsmyfile)

call wrtjobinfo(luo,0,r,s,c,nogrowth,smoothdelta,smoothxy,maxsmooth,optth,optwt,     &
                nodeth,npolygon,bndpolygon,                                          &
                rwkclrad,rwkclxc,rwkclyc,itmax,nbndnodes,nactnodes,nskiplowabs,      &
                pafecfile,resultsfile,joboptoutfile,joboptsmyfile,joboptpfcfile,     &
                chkdivergence,chkconvergence,datestamp,setmidsidenodes,              &
                title1,title2,title3,puppiesfile,minradcurvlim,programid,restorebnd, &
                constspacing,findcorners,nbnd,useglobalstress,useaveragedstress,     &
                usevonmisesstress,puppiesversion,closedbnd,multiplepeaks,stol,       &
                xeqn,yeqn,zeqn,optmodifyxyz,nuserparam,userparam,usemaxpospeaksth,   &
                useabsvaluesdr,bndlenfac)

write(luo,*)
call wrtmsgi(luo,'Iterations completed         = ',itnum)
call wrtmsgd(luo,'Elapsed time for job (hours) = ',elapsedtime)
write(luo,*)
write(luo,*) 'POSSIBLE LOCATION OF BEST SOLUTION:'
write(luo,*)
call wrtmsgdi(luo,'Minimum peak abs stress   = ', &
              minabsbndstrit, ' at iteration = ',iminabsbndstrit)
call wrtmsgdi(luo,'Minimum peak stress       = ', &
              minmaxbndstr,   ' at iteration = ',iminmaxbndstr)
call wrtmsgdi(luo,'Minimum range stress      = ', &
              minrngbndstr,   ' at iteration = ',iminrngbndstr)
call wrtmsgdi(luo,'Minimum range abs stress  = ', &
```

```
                minrngabsbndstr,' at iteration = ',iminrngabsbndstr)
        call wrtmsgdi(luo,'Maximum zone total length = ', &
                maxzonetotlen,  ' at iteration = ',imaxzonetotlen)
        write(luo,*)
        write(luo,*) 'PROGRESS SUMMARY:'
        write(luo,*)
 8      format(a8,a11,a9,2a11,a8,a11,<nzonesmax>(7a11))
        write(luo,8) 'IterNum','MaxAbsStr','%ofInit','RangeStr','MinRadCurv','NZones','TotalLen', &
                    (ZHstr(i,'Len'   ),ZHstr(i,'StrThr'),ZHstr(i,'PeakStr'),ZHstr(i,'MinStr'),   &
                     ZHstr(i,'AvgStr'),ZHstr(i,'RngStr'),ZHstr(i,'StdDev' ),i=1,nzonesmax)
        do j=1,itnum
 9       format(i8,f11.4,f9.3,2f11.4,i8,f11.4,<nzones(j)>(7f11.4))
         write(luo,9) j,maxabsbndstr(j),pcinit(j),rngbndstr(j),minradcurv(j),nzones(j),  &
                     zonetotlen(j),                                                      &
                     (zonet(i,j),sth(i,j),zonepeak(i,j),zonemin(i,j),zoneavg(i,j),      &
                     zonerange(i,j),zonesd(i,j),i=1,nzones(j))
        enddo
        close(luo)

        ! Write log of results to job output file.

        if (itnum == 1) then
          open (unit=luo,file=joboptoutfile)
          call wrtjobinfo(luo,0,r,s,c,nogrowth,smoothdelta,smoothxy,maxsmooth,optth,optwt,     &
                          nodeth,npolygon,bndpolygon,                                         &
                          rwkclrad,rwkclxc,rwkclyc,itmax,nbndnodes,nactnodes,nskiplowabs,     &
                          pafecfile,resultsfile,joboptoutfile,joboptsmyfile,joboptpfcfile,    &
                          chkdivergence,chkconvergence,datestamp,setmidsidenodes,             &
                          title1,title2,title3,puppiesfile,minradcurvlim,programid,restorebnd, &
                          constspacing,findcorners,nbnd,useglobalstress,useaveragedstress,    &
                          usevonmisesstress,puppiesversion,closedbnd,multiplepeaks,stol,      &
                          xeqn,yeqn,zeqn,optmodifyxyz,nuserparam,userparam,usemaxpospeaksth,  &
                          useabsvaluesdr,bndlenfac)

          write(luo,*)
          write(luo,*) 'ITERATION RESULTS:'
          write(luo,*)
        else
          open (unit=luo,file=joboptoutfile,access='append')
        endif

        call wrtmsgi(luo,'Iteration number        = ',itnum)
        call wrtmsgd(luo,'% of initial stress     = ',pcinit(itnum))
        call wrtmsgd(luo,'Stress threshold        = ',sth(1,itnum))
        call wrtmsgd(luo,'Max stress              = ',maxbndstr(itnum))
        call wrtmsgd(luo,'Min stress              = ',minbndstr(itnum))
        call wrtmsgd(luo,'Range stress            = ',rngbndstr(itnum))
        call wrtmsgd(luo,'Avg absolute stress     = ',avgabsbndstr(itnum))
        call wrtmsgd(luo,'Min radius of curvature = ',minradcurv(itnum))
        call wrtmsgi(luo,'Number passes rad curv  = ',nrcpass)
        call wrtmsgi(luo,'Number of stress zones  = ',nzones(itnum))
        write(luo,3)     'Peak stress in zone     = ',(zonepeak(i,itnum),i=1,nzones(itnum))
        write(luo,3)     'Minimum stress in zone  = ',(zonemin(i,itnum),i=1,nzones(itnum))
        write(luo,3)     'Average stress in zone  = ',(zoneavg(i,itnum),i=1,nzones(itnum))
        write(luo,3)     'Range stress in zone    = ',(zonerange(i,itnum),i=1,nzones(itnum))
        write(luo,3)     'Std dev stress in zone  = ',(zonesd(i,itnum),i=1,nzones(itnum))
        write(luo,5)     'Uniform zone length     = ',(zonet(i,itnum),i=1,nzones(itnum))
        call wrtmsgd(luo,'Total uniform length    = ',zonetotlen(itnum))
        call wrtmsgl(luo,'Polygon enforced        = ',wasconstrained)
        call wrtmsgd(luo,'Length moving boundary  = ',scurbndtotal*bndlenfac)
        call wrtmsgd(luo,'Elapsed time (hours)    = ',elapsedtime)

        write(luo,'(1x,a6,a5,a4,a5,a4,6a13,7a12,2a13,4a13,2a10,3a5,a8)') &
              'Node','Rest','Act','Smth','Bnd',                          &
              'Xdata','Ydata','Zdata','Xpafec','Ypafec','Zpafec',        &
              'BndStr','BigS1S2S3','Sigma1','Sigma2','Sigma3',           &
              'SigmaVM','DeltaStr','Drequest','Dactual',                 &
              'Xnew','Ynew','Xdiff','Ydiff',                             &
              'ThetaOut','RadOfCurv','Cncv','Zone','Insd','BndLenT'
        do j=1,nbnd
          do i=1,nbndnodes
            call dlocpt(xn(i,j,itnum),yn(i,j,itnum),xbndpolygon,ybndpolygon,npolygon,l,m)
            write(luo,'(1x,i6,i5,l4,i5,i4,6f13.7,7f12.5,2f13.8,4f13.7,f10.2,f10.3,3i5,f8.5)') &
              np(i,j,itnum),fixnodes(i),actnodes(i),smthnodes(i),j,                           &
              xr(i,j,itnum),yr(i,j,itnum),zr(i,j,itnum),                                      &
              xp(i,j,itnum),yp(i,j,itnum),zp(i,j,itnum),                                      &
              bndstr(i,itnum),bigs1s2s3(i,j,itnum),s1(i,j,itnum),s2(i,j,itnum),s3(i,j,itnum), &
```

```
      svm(i,j,itnum),deltabndstr(i,itnum),dr(i,itnum),da(i,itnum),                    &
      xn(i,j,itnum),yn(i,j,itnum),xd(i,itnum),yd(i,itnum),                            &
      theta(i,itnum)*radtodeg,radcurv(i,itnum),concave(i,itnum),izone(i),l,           &
      tcurbnd(i)/bndlenfac
  enddo
  if (closedbnd) then
    i=1
    call dlocpt(xn(i,j,itnum),yn(i,j,itnum),xbndpolygon,ybndpolygon,npolygon,l,m)
    write(luo,'(1x,i6,i5,l4,i5,i4,6f13.7,7f12.5,2f13.8,4f13.7,f10.2,f10.3,3i5,f8.5)') &
      np(i,j,itnum),fixnodes(i),actnodes(i),smthnodes(i),j,                           &
      xr(i,j,itnum),yr(i,j,itnum),zr(i,j,itnum),                                      &
      xp(i,j,itnum),yp(i,j,itnum),zp(i,j,itnum),                                      &
      bndstr(i,itnum),bigs1s2s3(i,j,itnum),s1(i,j,itnum),s2(i,j,itnum),s3(i,j,itnum), &
      svm(i,j,itnum),deltabndstr(i,itnum),dr(i,itnum),da(i,itnum),                    &
      xn(i,j,itnum),yn(i,j,itnum),xd(i,itnum),yd(i,itnum),                            &
      theta(i,itnum)*radtodeg,radcurv(i,itnum),concave(i,itnum),izone(i),l,           &
      1.0d0/bndlenfac
  endif
enddo
if (jobdiverging) then
  call wrtscnfile(luo,' ')
  call wrtscnfile(luo,'*** Divergence detected in shape optimisation job.')
  close(luo)
  goto 1060
endif
if (solved) then
  call wrtscnfile(luo,' ')
  call wrtscnfile(luo,'*** Shape optimisation job completed.')
  close(luo)
  goto 1060
endif
close(luo)

! Make a temporary copy of the PAFEC data file.

call fcopy(resultsfile,tempfile)

! Write new PAFEC data file with the new set of predicted nodal
! coordinates added to the end of the original nodal data listing.

open (unit=luj,file=resultsfile)
open (unit=lut,file=tempfile,iostat=ios,status='old')
if (ios /= 0) then
  write(*,*)
  write(*,*) '*** Could not open temporary file.'
  write(*,*) '*** tempfile=',tempfile(1:len_trim(tempfile))
  write(*,*)
  stop
endif
ios=0
pafblocksfound=.false.
pnpfin='C Predicted node positions from iteration number ='
do while (ios == 0)
  read(lut,'(a80)',iostat=ios,end=1050) aline
  if (itnum == 1) then
    if (index(aline,pnpfin(1:len_trim(pnpfin))) == 1) then
      ! Modify this line so that it does not affect the nodal updating procedure.
      write(*,*)
      write(*,*) '*** Found an existing set of predicted node positions.'
      write(*,*) '*** Modified text in line to continue with this job.'
      write(*,*)
      read(aline(index(aline,'=')+1:),*) ntemp
      write(luj,'(a,i6)') 'C Existing node locations at iteration number = ',ntemp
    else
      if (aline(1:1) == 'C' .or. aline(1:1) == 'c') then
        ! Do nothing because it's a comment line.
      else if (index(aline,'PAFBLOCKS') > 0 .and. &
               .not. pafblocksfound                ) then
        pafblocksfound=.true.
        call writenodes(luj,itnum,nbndnodes,nbnd,maxnode,maxbnd,maxit,np,xn,yn,zn,is2dgeom)
        if (nsupnodes > 0) then
          call writesupnodes(luj,itnum,nsupnodes,nsupbnd,maxnode,maxbnd,maxit,supnodes, &
                             xpsup,ypsup,zpsup,is2dgeom)
        endif
      endif
      write(luj,'(a)') aline(1:max(len_trim(aline),1))
    endif
```

```
            else
              if (index(aline,pnpfin(1:len_trim(pnpfin))) == 1) then
                do i=1,nbndnodes*nbnd+2
                  read(lut,*)
                enddo
                if (nsupnodes > 0) then
                  do i=1,nsupnodes*nsupbnd+3
                    read(lut,*)
                  enddo
                endif
                call writenodes(luj,itnum,nbndnodes,nbnd,maxnode,maxbnd,maxit,np,xn,yn,zn,is2dgeom)
                if (nsupnodes > 0) then
                  call writesupnodes(luj,itnum,nsupnodes,nsupbnd,maxnode,maxbnd,maxit,supnodes, &
                                     xpsup,ypsup,zpsup,is2dgeom)
                endif
              else
                write(luj,'(a)') aline(1:max(len_trim(aline),1))
              endif
            endif
          enddo
 1050     continue
          close(luj)
          close(lut)
          call fdelete(tempfile)

          ! Keep a record of node coordinates at each iteration by writing them to a file.

          if (itnum == 1) then
            open(unit=luo,file=joboptpfcfile)
            write(luo,'(a)') 'C '//programid(1:len_trim(programid))
            write(luo,'(a)') 'C'
            write(luo,'(a)') 'C Job start date: '//datestamp(1:len_trim(datestamp))
            write(luo,'(a)') 'C'
          else
            open(unit=luo,file=joboptpfcfile,access='append')
          endif
          call writenodes(luo,itnum,nbndnodes,nbnd,maxnode,maxbnd,maxit,np,xn,yn,zn,is2dgeom)
          if (nsupnodes > 0) then
            call writesupnodes(luo,itnum,nsupnodes,nsupbnd,maxnode,maxbnd,maxit,supnodes, &
                               xpsup,ypsup,zpsup,is2dgeom)
          endif
          close(luo)

        enddo

        maxitmsg='*** Maximum number of iterations has been reached.'
        call appendmsgtofile(luo,joboptoutfile,1,maxitmsg)
        call appendmsgtofile(luo,joboptsmyfile,1,maxitmsg)
        write(*,*) '***'
        write(*,*) maxitmsg(1:len_trim(maxitmsg))

        ! Cleanup temporary files.

 1060   continue
        write(*,*)
        call fclear(pupjnlfile)
        call fclear(pupoutfile)
        call fclear(bndnodesfile)
        stop

        end

!=======================================================================================================

        character*11 function ZHstr(n,str)

! Create a formatted string for the column heading for a given zone, using the supplied zone
! number and the initial part of the heading string.

        implicit none

        integer       n
        character*(*) str

        character*3   suffixstr
        character*11  fullstr
        character*11  blankstr
```

```
      integer       l,ls

      write(suffixstr,'(a1,i2.2)') 'Z',n

      fullstr=str(1:len_trim(str))//suffixstr

      ls=len(fullstr)
      l=len_trim(fullstr)
      blankstr=' '

      if (l < ls) fullstr = blankstr(1:ls-l)//fullstr

      ZHstr=fullstr

      return
      end

!=============================================================================================================

      subroutine writenodes(luo,itnum,nbndnodes,nbnd,maxnode,maxbnd, &
                            maxit,np,xn,yn,zn,is2dgeom)

      implicit none

      integer luo,itnum,nbnd,nbndnodes,maxnode,maxbnd,maxit
      integer np(maxnode,maxbnd,maxit)
      real*8  xn(maxnode,maxbnd,maxit)
      real*8  yn(maxnode,maxbnd,maxit)
      real*8  zn(maxnode,maxbnd,maxit)
      logical is2dgeom

      integer i,j

      write(luo,'(a,i6)') 'C Predicted node positions from iteration number = ',itnum
      write(luo,'(a)') 'C'
      do j=1,nbnd
        do i=1,nbndnodes
          if (is2dgeom) then
            write(luo,'(i7,2f15.7)') np(i,j,itnum),xn(i,j,itnum),yn(i,j,itnum)
          else
            write(luo,'(i7,3f15.7)') np(i,j,itnum),xn(i,j,itnum),yn(i,j,itnum),zn(i,j,itnum)
          endif
        enddo
      enddo
      write(luo,'(a)') 'C'

      return
      end

!=============================================================================================================

      subroutine writesupnodes(luo,itnum,nsupnodes,nsupbnd,maxnode,maxbnd, &
                               maxit,supnodes,xpsup,ypsup,zpsup,is2dgeom)

      implicit none

      integer luo,itnum,nsupbnd,nsupnodes,maxnode,maxbnd,maxit
      integer supnodes(maxnode,maxbnd)
      real*8  xpsup(maxnode,maxbnd,maxit)
      real*8  ypsup(maxnode,maxbnd,maxit)
      real*8  zpsup(maxnode,maxbnd,maxit)
      logical is2dgeom

      integer i,j

      write(luo,'(a,i6)') 'C Supplementary nodes.'
      write(luo,'(a)') 'C'
      do j=1,nsupbnd
        do i=1,nsupnodes
          if (is2dgeom) then
            write(luo,'(i7,2f15.7)') supnodes(i,j),xpsup(i,j,itnum),ypsup(i,j,itnum)
          else
            write(luo,'(i7,3f15.7)') supnodes(i,j),xpsup(i,j,itnum),ypsup(i,j,itnum),zpsup(i,j,itnum)
          endif
        enddo
      enddo
      write(luo,'(a)') 'C'
```

```
      return
      end

!=======================================================================================================

      integer function lenstr(s)

! Determine the length ls of a string s.

      implicit none

      character s*(*)  ! String.

      integer   ls     ! Number of characters in the string s.
      integer   i

      ls = len(s)

      if (ls == 0) goto 20

      ! Determine the location of the rightmost non-blank character
      ! and use this to define the length of the string.

10    if (s(ls:ls) == ' ') then
        ls = ls - 1
        if (ls > 0) goto 10
      endif

      ! Check to see if a null character is present, and adjust the
      ! length of the string to exclude this.

      do i=1,ls
        if (ichar(s(i:i)) == 0) then
          ls = i - 1
          goto 20
        endif
      enddo

20    continue

      lenstr = ls

      return
      end

!=======================================================================================================

      subroutine wrtmsgl(lu,msg,l)

      implicit none

      integer   lu
      character msg*(*)
      logical   l

      if (l) then
        write(lu,'(1x,a,a13)') msg,' Yes'
      else
        write(lu,'(1x,a,a13)') msg,' No'
      endif

      return
      end

!=======================================================================================================

      subroutine wrtmsgi(lu,msg,i)

      implicit none

      integer   lu
      character msg*(*)
      integer   i

      write(lu,'(1x,a,i13)') msg,i
```

```
      return
      end

!=====================================================================================================

      subroutine wrtmsgd(lu,msg,d)

      implicit none

      integer   lu
      character msg*(*)
      real*8    d

      write(lu,'(1x,a,f13.6)') msg,d

      return
      end

!=====================================================================================================

      subroutine wrtmsga(lu,msg,a,n)

      implicit none

      integer   lu
      character msg*(*)
      real*8    a(*)
      integer   n

      integer   i

      write(lu,'(1x,a,f13.6)') msg,(a(i),i=1,n)

      return
      end

!=====================================================================================================

      subroutine wrtmsgz(lu,msg,z,n)

      implicit none

      integer    lu,n
      character  msg*(*)
      complex*16 z(n)

      integer    i
      character  b*(len(msg))

      b=' '

      write(lu,'(1x,a,''('',f12.3,'','',f12.3,'')'')') msg,z(1),(b,z(i),i=2,n)

      return
      end

!=====================================================================================================

      subroutine wrtmsgc(lu,msg,c)

      implicit none

      integer   lu
      character msg*(*)
      character c*(*)

      if (len(c) > 0) then
        write(lu,'(1x,a,a)') msg,c(1:len_trim(c))
      else
        write(lu,'(1x,a,a)') msg
      endif

      return
      end

!=====================================================================================================
```

```
      subroutine wrtmsgdi(lu,msgd,d,msgi,i)

      implicit none

      integer   lu
      character msgd*(*),msgi*(*)
      real*8    d
      integer   i

      write(lu,'(1x,a,f13.6,a,i)') msgd,d,msgi,i

      return
      end

!=====================================================================================================

      subroutine wrtscnfile(lu,msg)

      implicit none

      integer   lu
      character msg*(*)

      integer   l

      l=len_trim(msg)

      write(lu,'(1x,a)') msg(1:l)
      write(* ,'(1x,a)') msg(1:l)

      return
      end

!=====================================================================================================

      subroutine appendmsgtofile(lu,outfile,nskip,msg)

      integer   lu
      character outfile*(*)
      integer   nskip
      character msg*(*)

      open(unit=lu,file=outfile,access='append')
      do i=1,nskip
        write(lu,*)
      enddo
      write(lu,'(1x,a)') msg(1:len_trim(msg))
      close(lu)

      return
      end

!=====================================================================================================

      subroutine lbstrip(s,ls)

! This subroutine strips leading blanks from a string s, and returns
! the length of the string ls.

      implicit none

      character s*(*)
      integer   ls

      integer   iloc
      integer   i

      ! Determine the length of the string and clip any extra characters.

      ls=len_trim(s)

      if (ls == 0) return

      s = s(1:ls)

      ! Determine first non-blank character in the string.
```

```
      do i = 1,ls
        if (s(i:i) /= ' ') then
          iloc = i
          goto 10
        endif
      enddo

   10 if (iloc > 1) then
        s = s(iloc:ls)
        ls = len_trim(s)
      endif

      return
      end

!=====================================================================================================

      logical function fexist(s)

      implicit none

      character s*(*)

      logical   ex

      inquire(file=s,exist=ex)

      fexist=ex

      return
      end

!=====================================================================================================

      subroutine syscmd(cmd)

      implicit none

      character cmd*(*)

      call system(cmd)

      return
      end

!=====================================================================================================

      subroutine findstr(lui,linestr,substr,ipos)

! Scans the strings in a file to find a substring substr. Returns ipos
! greater than zero if the substring was found.

      implicit none

      integer   lui
      integer   ipos
      character substr*(*)
      character linestr*(*)

      integer   ios

      integer   index

      ipos=0
      ios=0

      do while (ipos == 0 .and. ios == 0)
        read(lui,'(a)',iostat=ios) linestr
        if (ios == 0) then
          ipos=index(linestr,substr)
        endif
      enddo

      return
      end

!=====================================================================================================
```

```
      subroutine fdelete(fname)

      implicit none

      character fname*(*)

      call syscmd('rm '//fname)

      return
      end

!=====================================================================================================

      subroutine fclear(fname)

      implicit none

      character fname*(*)
      logical   fexist

      if (fexist(fname)) call syscmd('rm '//fname)

      return
      end

!=====================================================================================================

      subroutine fmove(fsrc,ftar)

      implicit none

      character fsrc*(*),ftar*(*)

      call syscmd('mv '//fsrc//' '//ftar)

      return
      end

!=====================================================================================================

      subroutine fcopy(fsrc,ftar)

      implicit none

      character fsrc*(*),ftar*(*)

      call syscmd('cp '//fsrc//' '//ftar)

      return
      end

!=====================================================================================================

      subroutine wrtpupjnlver(filename,filejob,filebnd,puppiesversion, &
                              useglobalstress,useaveragedstress)

! Create a PUPPIES journal file that will be used to extract out the
! nodal coordinates and the principal stresses at those same nodes.
! filename is string with the name of the PUPPIES journal file,
! filejob is a string with the name of the PAFEC job whose output is
! to be processed, and filebnd is a string with the name of the file
! containing the list of nodes.

      implicit none

      character filename*(*),filejob*(*),filebnd*(*),puppiesversion*(*)
      logical   useglobalstress,useaveragedstress

      open(unit=20,file=filename)

      write(20,'(a)') puppiesversion
      write(20,'(a)') filejob
      write(20,'(a)') 'Y'
      write(20,'(a)') '        -1'
      write(20,'(a)') filebnd
      write(20,'(a)') ' '
```

```
      write(20,'(a)') '         1'
      write(20,'(a)') '         1'
      write(20,'(a)') '         3'
      write(20,'(a)') '         1'
      write(20,'(a)') '         1'
      write(20,'(a)') '         0'
      write(20,'(a)') '         0'
      write(20,'(a)') '         0'
      write(20,'(a)') '         6'
      write(20,'(a)') '         1'
      write(20,'(a)') '         1'
      write(20,'(a)') '         2'
      if (useaveragedstress) then
        write(20,'(a)') '        -7'
      endif
      if (useglobalstress) then
        write(20,'(a)') '         2'
      endif
      write(20,'(a)') '         3'
      write(20,'(a)') 'Y'
      write(20,'(a)') '         0'
      write(20,'(a)') '         0'
      write(20,'(a)') '         0'
      write(20,'(a)') '         0'

      close(20)

      return
      end

!=======================================================================================================

      subroutine wrtbndnodes(fname,maxnode,nbnd,nbndnodes,bndnodes)

      implicit none

      character fname*(*)
      integer   maxnode,nbnd,nbndnodes,bndnodes(maxnode,nbnd)

      integer   i,j

      open(unit=20,file=fname)

      do j=1,nbnd
        do i=1,nbndnodes
          write(20,*) bndnodes(i,j)
        enddo
      enddo

      close(20)

      return
      end

!=======================================================================================================

      subroutine getstresspup(lup,puppiesfile,pupjnlfile,resultsname,bndnodesfile,          &
                              pupoutfile,joboptscnfile,puppiesversion,useglobalstress,      &
                              useaveragedstress,scn,stopafterpuppies,skipstressextraction,  &
                              itnum,maxit,maxnode,maxbnd,nbnd,nbndnodes,bndnodes,           &
                              np,xp,yp,zp,s1,s2,s3,svm)

! Use PUPPIES to get the required geometry data and stress data at the specified nodes.

      implicit none

      integer   lup,maxit,maxbnd,maxnode,nbnd,nbndnodes,itnum
      integer   np(maxnode,maxbnd,itnum),bndnodes(maxnode,maxbnd)
      real*8    xp(maxnode,maxbnd,maxit),yp(maxnode,maxbnd,maxit),zp(maxnode,maxbnd,maxit)
      real*8    s1(maxnode,maxbnd,maxit),s2(maxnode,maxbnd,maxit),s3(maxnode,maxbnd,maxit)
      real*8    svm(maxnode,maxbnd,maxit)
      character puppiesfile*(*),pupjnlfile*(*),resultsname*(*),bndnodesfile*(*)
      character pupoutfile*(*),puppiesversion*(*),scn*(*),joboptscnfile*(*)
      logical   useglobalstress,useaveragedstress,stopafterpuppies,skipstressextraction

      integer   ios,i,j,nnodestot,ielem
      real*8    sxx,syy,szz
```

```
real*8    txy,tyz,tzx
real*8    s11,s22
real*8    a11,a22,a33
real*8    rnum

logical fexist

if (len_trim(puppiesfile) > 0) then
  if (fexist(puppiesfile)) then
    write(*,*) '***'
    write(*,*) '*** Running user selected PUPPIES: ',puppiesfile(1:len_trim(puppiesfile))
    write(*,*) '*** PUPPIES version = ',puppiesversion
    call wrtpupjnlver(pupjnlfile,resultsname,bndnodesfile,puppiesversion, &
                      useglobalstress,useaveragedstress)
    call wrtbndnodes(bndnodesfile,maxnode,nbnd,nbndnodes,bndnodes)
    call syscmd(puppiesfile(1:len_trim(puppiesfile)+1)//pupjnlfile(1:len_trim(pupjnlfile))//scn)
    call fclear(joboptscnfile(1:len_trim(joboptscnfile)))
  else
    write(*,*) '***'
    write(*,*) '*** Custom version of PUPPIES not found.'
    write(*,*) '*** puppiesfile=',puppiesfile(1:len_trim(puppiesfile))
    write(*,*)
    stop
  endif
else
  ! Use the PUPPIES version that is installed on the system.
  write(*,*) '*** Running standard PUPPIES installation.'
  write(*,*) '*** PUPPIES version = ','@8.14'
  call wrtpupjnlver(pupjnlfile,resultsname,bndnodesfile,'@8.14', &
                    useglobalstress,useaveragedstress)
  call wrtbndnodes(bndnodesfile,maxnode,nbnd,nbndnodes,bndnodes)
  call syscmd('pup '//pupjnlfile(1:len_trim(pupjnlfile))//scn)
  call fclear(joboptscnfile(1:len_trim(joboptscnfile)))
endif

if (stopafterpuppies) then
  write(*,*)
  write(*,*) '*** User request to stop after PUPPIES is run.'
  write(*,*)
  stop
endif

! Open the PUPPIES output file and read through text lines to find the
! first line of numeric data that contains the nodal coordinates.

open(unit=lup,file=pupoutfile,iostat=ios,status='old')
if (ios /= 0) then
  write(*,*)
  write(*,*) '*** Could not open PUPPIES output file.'
  write(*,*) '*** pupoutfile=',pupoutfile(1:len_trim(pupoutfile))
  write(*,*)
  stop
endif

ios=1
do while (ios /= 0)
  read(lup,*,iostat=ios) rnum
enddo
backspace(lup)

! Read in the node numbers and their nodal coordinates.

nnodestot=0
ios=0
j=0
do while (ios == 0 .and. j < nbnd)
  j=j+1
  i=0
  do while (ios == 0 .and. i < nbndnodes)
    i=i+1
    np(i,j,itnum)=0
    read(lup,*,iostat=ios) np(i,j,itnum),xp(i,j,itnum),yp(i,j,itnum),zp(i,j,itnum)
    if (ios == 0) nnodestot=nnodestot+1
  enddo
enddo

if (ios /= 0) then
```

```
      write(*,*)
      write(*,*) '*** Error while reading node coordinates from '// &
                 pupoutfile(1:len_trim(pupoutfile))//'.'
      write(*,*) '***',nnodestot,'nodes were read from the file.'
      write(*,*)
      stop
    endif

    if (nnodestot == 0) then
      write(*,*)
      write(*,*) '*** No nodes were read from '//pupoutfile(1:len_trim(pupoutfile))//'.'
      write(*,*)
      stop
    else
      write(*,*) '***'
      write(*,*) '***',nnodestot,'nodes were read from '//pupoutfile(1:len_trim(pupoutfile))//'.'
      write(*,*) '***'
    endif

    if (nnodestot /= nbndnodes*nbnd) then
      write(*,*)
      write(*,'(1x,a)') '*** Number of nodes read from PUPPIES output file does not'
      write(*,'(1x,a)') '*** match the number of boundary nodes that were defined.'
      write(*,*)
      stop
    endif

    if (.not. skipstressextraction) then

      ! Read through more lines of text to find the first line of stress data.

      ios=1
      do while (ios /= 0)
        read(lup,*,iostat=ios) rnum
      enddo
      backspace(lup)

      ! Read in principal stresses and associated angles at each of the nodes.

      if (useglobalstress) then
        do j=1,nbnd
          do i=1,nbndnodes
            read(lup,*) ielem,np(i,j,itnum),sxx,syy,szz,txy,tyz,tzx,svm(i,j,itnum)
            ! Compute the principal stresses in 2D from the global stresses.
            call s1s2th1th2(sxx,syy,txy,s11,s22,a11,a22)
            s1(i,j,itnum)=s11
            s2(i,j,itnum)=s22
            s3(i,j,itnum)=0.0d0
          enddo
        enddo
      else
        do j=1,nbnd
          do i=1,nbndnodes
            read(lup,*) np(i,j,itnum),s1(i,j,itnum),s2(i,j,itnum),s3(i,j,itnum), &
                        a11,a22,a33,svm(i,j,itnum)
          enddo
        enddo
      endif

    endif

    close(lup)

    return
    end

!=====================================================================================================

    subroutine wrtjobinfo(lu,nb,r,s,c,nogrowth,smoothdelta,smoothxy,maxsmooth,optth,optwt,     &
                          nodeth,npolygon,bndpolygon,                                         &
                          rwkclrad,rwkclxc,rwkclyc,itmax,nbndnodes,nactnodes,nskiplowabs,     &
                          pafecfile,resultsfile,joboptoutfile,joboptsmyfile,joboptpfcfile,    &
                          chkdivergence,chkconvergence,datestamp,setmidsidenodes,             &
                          title1,title2,title3,puppiesfile,minradcurvlim,programid,restorebnd, &
                          constspacing,findcorners,nbnd,useglobalstress,useaveragedstress,    &
                          usevonmisesstress,puppiesversion,closedbnd,multiplepeaks,stol,      &
                          xeqn,yeqn,zeqn,optmodifyxyz,nuserparam,userparam,usemaxpospeaksth,  &
```

```
                        useabsvaluesdr,bndlenfac)

      implicit none

      integer    lu,nb
      real*8     r,s,c
      logical    nogrowth,smoothdelta,smoothxy
      integer    maxsmooth,optth,optwt,nodeth,npolygon
      complex*16 bndpolygon(npolygon)
      real*8     rwkclrad,rwkclxc,rwkclyc
      integer    itmax,nbndnodes,nactnodes,nskiplowabs
      character  pafecfile*(*),resultsfile*(*),joboptoutfile*(*)
      character  joboptsmyfile*(*),joboptpfcfile*(*)
      logical    chkdivergence,chkconvergence
      character  datestamp*(*)
      logical    setmidsidenodes
      character  title1*(*),title2*(*),title3*(*)
      character  puppiesfile*(*)
      real*8     minradcurvlim
      character  programid*(*)
      logical    restorebnd,constspacing,findcorners
      integer    nbnd
      logical    useglobalstress,useaveragedstress,usevonmisesstress
      character  puppiesversion*(*)
      logical    closedbnd,multiplepeaks
      real*8     stol
      character  xeqn*(*),yeqn*(*),zeqn*(*)
      integer    optmodifyxyz
      integer    nuserparam
      real*8     userparam(*)
      logical    usemaxpospeaksth
      logical    useabsvaluesdr
      real*8     bndlenfac

      integer   i,lentitle1,lentitle2,lentitle3

      lentitle1=len_trim(title1)
      lentitle2=len_trim(title2)
      lentitle3=len_trim(title3)

      do i=1,nb
        call wrtmsgc(lu,'','')
      enddo

      call wrtmsgc(lu,programid(1:len_trim(programid)),'')
      call wrtmsgc(lu,'','')
      call wrtmsgc(lu,'Job start date: ',datestamp)
      call wrtmsgc(lu,'','')
      if (lentitle1 > 0 .or. lentitle2 > 0 .or. lentitle3 > 0) then
        if (lentitle1 > 0) call wrtmsgc(lu,title1(1:lentitle1),'')
        if (lentitle2 > 0) call wrtmsgc(lu,title2(1:lentitle2),'')
        if (lentitle3 > 0) call wrtmsgc(lu,title3(1:lentitle3),'')
        call wrtmsgc(lu,'','')
      endif
      call wrtmsgc(lu,'JOB PARAMETERS FOR OPTIMISATION RUN:','')
      call wrtmsgc(lu,'','')
      call wrtmsgc(lu,'Initial PAFEC data file      = ',pafecfile)
      call wrtmsgc(lu,'Iterated PAFEC data file     = ',resultsfile)
      call wrtmsgc(lu,'Optimisation summary file    = ',joboptsmyfile)
      call wrtmsgc(lu,'Optimisation output file     = ',joboptoutfile)
      call wrtmsgc(lu,'Optimisation nodes file      = ',joboptpfcfile)
      if (len_trim(puppiesfile) > 0) then
        call wrtmsgc(lu,'User-defined PUPPIES program = ',puppiesfile)
      endif
      call wrtmsgc(lu,'Version of PUPPIES used      = ',puppiesversion)
      if (len_trim(xeqn) > 0) call wrtmsgc(lu,'Equation for x ordinate      = ',xeqn)
      if (len_trim(yeqn) > 0) call wrtmsgc(lu,'Equation for y ordinate      = ',yeqn)
      if (len_trim(zeqn) > 0) call wrtmsgc(lu,'Equation for z ordinate      = ',zeqn)
      call wrtmsgd(lu,'Characteristic length        = ',r)
      call wrtmsgd(lu,'Step size scaling factor     = ',s)
      call wrtmsgd(lu,'Convergence criterion        = ',c)
      call wrtmsgd(lu,'Stress uniformity tolerance  = ',stol)
      call wrtmsgd(lu,'Boundary length factor       = ',bndlenfac)
      call wrtmsgd(lu,'Min rad of curvature limit   = ',minradcurvlim)
      if (rwkclrad > 0.0d0) then
        call wrtmsgd(lu,'Rework circle: radius        = ',rwkclrad)
        call wrtmsgd(lu,'Rework circle: x centre      = ',rwkclxc)
```

```
        call wrtmsgd(lu,'Rework circle: y centre      = ',rwkclyc)
      endif
      call wrtmsgl(lu,'Check solution divergence    = ',chkdivergence)
      call wrtmsgl(lu,'Check solution convergence   = ',chkconvergence)
      call wrtmsgl(lu,'No growth                    = ',nogrowth)
      call wrtmsgl(lu,'Maintain element spacing     = ',constspacing)
      call wrtmsgl(lu,'Smooth delta                 = ',smoothdelta)
      call wrtmsgl(lu,'Smooth xy                    = ',smoothxy)
      call wrtmsgi(lu,'No smoothing after iteration = ',maxsmooth)
      call wrtmsgl(lu,'Set PAFBLOCK midside nodes   = ',setmidsidenodes)
      if (nodeth > 0) then
        call wrtmsgi(lu,'Stress threshold node        = ',nodeth)
      else
        call wrtmsgi(lu,'Stress threshold calculation = ',optth)
      endif
      call wrtmsgi(lu,'Weighting calculation        = ',optwt)
      call wrtmsgl(lu,'Use max positive peak sth    = ',usemaxpospeaksth)
      call wrtmsgl(lu,'Use abs values to calc dr    = ',useabsvaluesdr)
      call wrtmsgl(lu,'Use von Mises stress         = ',usevonmisesstress)
      call wrtmsgl(lu,'Use global stresses          = ',useglobalstress)
      call wrtmsgl(lu,'Use averaged nodal stresses  = ',useaveragedstress)
      call wrtmsgi(lu,'Threshold avg skips low abs  = ',nskiplowabs)
      call wrtmsgi(lu,'Maximum iterations allowed   = ',itmax)
      call wrtmsgi(lu,'Modify (x,y,z) option        = ',optmodifyxyz)
      call wrtmsgi(lu,'Number of boundary nodes     = ',nbndnodes)
      call wrtmsgi(lu,'Number of active nodes       = ',nactnodes)
      call wrtmsgl(lu,'Closed boundary              = ',closedbnd)
      call wrtmsgl(lu,'Use multiple peaks           = ',multiplepeaks)
      call wrtmsgl(lu,'Restore boundary curvature   = ',restorebnd)
      call wrtmsgl(lu,'Find corners and adjust mesh = ',findcorners)
      call wrtmsgi(lu,'Number of boundary lines     = ',nbnd)
      call wrtmsgi(lu,'Number of polygon points     = ',npolygon)
      call wrtmsgz(lu,'Polygon boundary points      = ',bndpolygon,npolygon)
      if (nuserparam > 0) then
        call wrtmsgi(lu,'Number user-defined params   = ',nuserparam)
        call wrtmsga(lu,'User-defined params          = ',userparam,nuserparam)
      endif

      return
      end

!=======================================================================================================

      subroutine copyfile80(source,target)

      character source*(*),target*(*)

      character aline*80
      integer ios

      open(unit=20,file=source,err=1010,status='old')
      open(unit=21,file=target)
      ios=0
      do while (ios == 0)
        read(20,'(a80)',iostat=ios,end=1000) aline
        write(21,'(a)') aline(1:len_trim(aline))
      enddo
 1000 continue
      close (20)
      close (21)
      return

 1010 write(*,*) '*** Could not open '//source(1:len_trim(source))//'.'
      stop

      end

!=======================================================================================================

      subroutine lagrange(tp,fp,dfp,t,f)

! Interpolation and calculation of first derivative using a Lagrange
! second-order polynomial approximation.

      real*8 tp,fp,dfp,t(3),f(3)
      real*8 a1,a2,a3
```

```
      a1=(t(1)-t(2))*(t(1)-t(3))
      a2=(t(2)-t(1))*(t(2)-t(3))
      a3=(t(3)-t(1))*(t(3)-t(2))

      fp = (tp-t(2))*(tp-t(3))/a1*f(1) + &
           (tp-t(1))*(tp-t(3))/a2*f(2) + &
           (tp-t(1))*(tp-t(2))/a3*f(3)

      dfp = ((tp-t(2))+(tp-t(3)))/a1*f(1) + &
            ((tp-t(1))+(tp-t(3)))/a2*f(2) + &
            ((tp-t(1))+(tp-t(2)))/a3*f(3)

      return
      end

!=====================================================================================================

      subroutine dabsord(a,n)

! dabsord is used to reorder the elements of the double precision
! array a so that abs(a(i)) <= abs(a(i+1)) for i = 1,...,n-1.
! It is assumed that n >= 1.
!
! From NSWC FORTRAN Subroutine Library.

      double precision a(n), s
      integer k(10)

      data k(1)/1/, k(2)/4/, k(3)/13/, k(4)/40/, k(5)/121/, k(6)/364/, &
           k(7)/1093/, k(8)/3280/, k(9)/9841/, k(10)/29524/

      ! Selection of the increments k(i) = (3**i-1)/2.

      if (n  <  2) return
      imax = 1
      do 10 i = 3,10
        if (n <= k(i)) go to 20
        imax = imax + 1
   10 continue

      ! Stepping through the increments k(imax),...,k(1).

   20 i = imax
      do 40 ii = 1,imax
        ki = k(i)

      ! Sorting elements that are ki positions apart
      ! so that abs(a(j)) <= abs(a(j+ki)).

        jmax = n - ki
        do 31 j = 1,jmax
          l = j
          ll = j + ki
          s = a(ll)
   30     if (abs(s) >= abs(a(l))) go to 31
            a(ll) = a(l)
            ll = l
            l = l - ki
            if (l  >  0) go to 30
   31    a(ll) = s

   40 i = i - 1

      return
      end

!=====================================================================================================

      subroutine dcopyvec(source,target,ibeg,iend)

! Copy some elements from one vector to another.

      real*8  source(*),target(*)
      integer ibeg,iend

      integer i
```

```
      do i=ibeg,iend
        target(i-ibeg+1)=source(i)
      enddo

      return
      end

!=====================================================================================================

      subroutine dabsvec(v,n)

! Take absolute value of each element in the vector v.

      real*8  v(n)
      integer n

      integer i

      do i=1,n
        v(i)=abs(v(i))
      enddo

      return
      end

!=====================================================================================================

      real*8 function dsumvec(v,n)

! Compute the sum of the values of the elements in v.

      real*8  v(n)
      integer n

      real*8  s
      integer i

      s=0.0d0
      do i=1,n
        s=s+v(i)
      enddo

      dsumvec=s

      return
      end

!=====================================================================================================

      real*8 function davgvec(v,n)

! Compute average of value of the elements in v.

      real*8  v(n)
      integer n

      real*8  s
      integer i

      s=0.0d0

      do i=1,n
        s=s+v(i)
      enddo

      davgvec=s/n

      return
      end

!=====================================================================================================

      integer function imaxvec(ivec,n)

! Get the maximum of the n values stored in an integer vector ivec.

      implicit none
```

```
      integer ivec(*),n

      integer i,m

      m=ivec(1)

      do i=2,n
        m=max(m,ivec(i))
      enddo

      imaxvec=m

      return
      end

!=====================================================================================================

      real*8 function dmaxvec(v,n)

! Get the maximum of the n values stored in a vector v(1)...v(n).

      implicit none

      real*8  v(*)
      integer n

      integer i
      real*8  maxval

      maxval=v(1)

      do i=2,n
        maxval=max(maxval,v(i))
      enddo

      dmaxvec=maxval

      return
      end

!=====================================================================================================

      subroutine intersectlinesphere(x1,y1,z1,x2,y2,z2,r,xc,yc,zc, &
                                     ni,xi1,yi1,zi1,xi2,yi2,zi2)

! Compute the intersection points for a line defined by the two points
! (x1,y1,z1) and (x2,y2,z2) with the sphere whose equation is:
!
!  (x-xc)**2 + (y-yc)**2 + (z-zc)**2 = r**2.
!
! The formulas used here were obtained from Paul Bourke's web page:
!
! http://www/swin.edu.au/astronomy/pbourke/geometry/sphereline/index.html

      implicit none

      real*8  x1,y1,z1,x2,y2,z2,r,xc,yc,zc,xi1,yi1,zi1,xi2,yi2,zi2
      integer ni

      real*8  a,b,c,d,u1,u2

      a=(x2-x1)**2+(y2-y1)**2+(z2-z1)**2
      b=2.0d0*((x2-x1)*(x1-xc)+(y2-y1)*(y1-yc)+(z2-z1)*(z1-zc))
      c=xc**2+yc**2+zc**2+x1**2+y1**2+z1**2-2*(xc*x1+yc*y1+zc*z1)-r**2

      d=b**2-4.0d0*a*c

      if (d < 0.0d0) then       ! No intersection with sphere.
        ni=0
      else if (d == 0.0d0) then  ! Line is tangent to sphere.
        ni=1
        u1=-b/(2*a)
        xi1=x1+u1*(x2-x1)
        yi1=y1+u1*(y2-y1)
        zi1=z1+u1*(z2-z1)
      else if (d > 0.0d0) then  ! Line intersects with sphere.
```

```
        ni=2
        u1=(-b+sqrt(d))/(2*a)
        u2=(-b-sqrt(d))/(2*a)
        xi1=x1+u1*(x2-x1)
        yi1=y1+u1*(y2-y1)
        zi1=z1+u1*(z2-z1)
        xi2=x1+u2*(x2-x1)
        yi2=y1+u2*(y2-y1)
        zi2=z1+u2*(z2-z1)
      endif

      return
      end

!=============================================================================================================

      subroutine rotationdir(xc,yc,x1,y1,x2,y2,cwise)

! Given the coordinates of the centre of a circle (xc,yc), and two
! points on the circle (x1,y1) and (x2,y2), this subprogram returns:
!
! cwise =  1 if going from point 1 to point 2 is in the clockwise sense.
! cwise = -1 if going from point 1 to point 2 is in the anticlockwise sense.
! cwise =  0 if the points are coincident.

      implicit none

      real*8  xc,yc,x1,y1,x2,y2
      integer cwise

      real*8  a

      a=(x1-xc)*(y2-yc)-(y1-yc)*(x2-xc)

      if (a == 0.0d0) then
        cwise=0
      else if (a < 0.0d0) then
        cwise=+1
      else
        cwise=-1
      endif

      return
      end

!=============================================================================================================

      subroutine rotdirection(xo,yo,x1,y1,x2,y2,cwise)

! Given the coordinates of the centre of a circle (xo,yo), and two
! points on the circle (x1,y1) and (x2,y2), this subprogram returns:
!
! cwise =  1 if going from point 1 to point 2 is in the clockwise sense.
! cwise = -1 if going from point 1 to point 2 is in the anticlockwise sense.
! cwise =  0 if the points are coincident.
!
! Algorithm:
!
! We start with the formula for the area of triangle ABC
! with coordinates of the vertices (xa,ya), (xb,yb) and (xc,yc):
!
!                |1     1     1|
!    Area = (1/2)|xa   xb    xc|
!                |ya   yb    yc|
!
! and we have that
!
!   |1    1   1|   | a11 a12 a13 |   | 1  1  1  |
!   |xa  xb  xc| = | a21 a22 a23 | = | xo x1 x2 |
!   |ya  yb  yc|   | a31 a32 a33 |   | yo y1 y2 |
!
! The determinant is positive if the points A, B, C are taken in an
! anticlockwise direction.

      implicit none

      real*8  xo,yo,x1,y1,x2,y2
```

```
      integer cwise

      real*8  a11,a12,a13,a21,a22,a23,a31,a32,a33,det

      a11=1.0d0
      a12=1.0d0
      a13=1.0d0
      a21=xo
      a22=x1
      a23=x2
      a31=yo
      a32=y1
      a33=y2

      det=a11*(a22*a33-a32*a23)-a12*(a21*a33-a31*a23)+a13*(a21*a32-a31*a22)

      if (det > 0.0d0) then
        cwise=-1
      else if (det < 0.0d0) then
        cwise=+1
      else
        cwise=0
      endif

      return
      end

!=====================================================================================================

      subroutine leftright(xa,ya,x1,y1,x2,y2,onleft)

! Given the coordinates of a point A (xa,ya), and two points that
! define a line (x1,y1) and (x2,y2), this subprogram returns:
!
! onleft =  1 if when going from point 1 to point 2 the point A is on the left.
! onleft = -1 if when going from point 1 to point 2 the point A is on the right.
! onleft =  0 if point A is colinear with the line between points 1 and 2.

      implicit none

      real*8  xa,ya,x1,y1,x2,y2
      integer onleft

      real*8  c

      c=(x2-x1)*(ya-y1)-(y2-y1)*(xa-x1)

      if (c == 0.0d0) then
        onleft=0
      else if (c > 0.0d0) then
        onleft=+1
      else
        onleft=-1
      endif

      return
      end

!=====================================================================================================

      subroutine circle(x1,y1,x2,y2,x3,y3,xc,yc,r,det)

! This subprogram finds the centre coordinates (xc,yc) and radius r
! of a circle passing through three points, (x1,y1) (x2,y2) (x3,y3).
!
! If the value of det is zero, then a circle through the three given
! points does not exist. e.g. the three points are colinear.

      implicit none

      real*8 x1,y1,x2,y2,x3,y3,xc,yc,r,det

      real*8 a1,a2,c1,c2,b1,b2

      a1=2.0d0*(x2-x1)
      a2=2.0d0*(x3-x2)
      b1=2.0d0*(y2-y1)
```

```
      b2=2.0d0*(y3-y2)
      c1=x2**2+y2**2-x1**2-y1**2
      c2=x3**2+y3**2-x2**2-y2**2

      det=a1*b2-b1*a2

      if (det /= 0.0d0) then
        xc=(c1*b2-b1*c2)/det
        yc=(a1*c2-c1*a2)/det
        a1=xc-x1
        a2=yc-y1
        r=dsqrt(a1**2+a2**2)
      else
        xc=0.0d0
        yc=0.0d0
        r=0.0d0
      endif

      return
      end

!=====================================================================================================

      subroutine settomid(x1,y1,x2,y2,x3,y3)

      implicit none

      real*8 x1,y1,x2,y2,x3,y3

      real*8 xc,yc,r,det,th1,th2,th3

      ! Determine circle fit (if valid for these three points).

      call circle(x1,y1,x2,y2,x3,y3,xc,yc,r,det)

      ! Adjust location of second point to be in the middle.

      if (det == 0.0d0) then
        ! The 3 points lie on a straight line.
        x2=(x1+x3)/2.0d0
        y2=(y1+y3)/2.0d0
      else
        ! The 3 points lie on a circle.
        th1=atan2(y1-yc,x1-xc)
        th3=atan2(y3-yc,x3-xc)
        th2=(th1+th3)/2.0d0
        x2=xc+r*cos(th2)
        y2=yc+r*sin(th2)
      endif

      return
      end

!=====================================================================================================

      subroutine radcurvcirc(n,x,y,xmirnodes,ymirnodes,radcurv,maxrad, &
                             clockwise,concave,elementsonright)

! Compute the radius of curvature at the (x(i),y(i)) coordinates, using
! axis of symmetry relationships if required. The maximum radius of
! curvature to be returned is maxrad, to allow for the case where any
! given three points are colinear. This would produce an infinite
! radius of curvature. The radius of curvature is computed by fitting
! a circle through three points at a time while moving along the
! supplied boundary.

      implicit none

      integer n,clockwise(*),concave(*)
      real*8  x(*),y(*),radcurv(*),maxrad
      logical xmirnodes(*),ymirnodes(*),elementsonright

      integer i
      real*8  det,x1,y1,x2,y2,x3,y3,xc,yc
      logical straightline

      do i=1,n
```

```
          if (i == 1) then
            if (xmirnodes(i)) then
              x1=-x(i+1)
              y1=y(i+1)
              x2=x(i)
              y2=y(i)
              x3=x(i+1)
              y3=y(i+1)
            else if (ymirnodes(i)) then
              x1=x(i+1)
              y1=-y(i+1)
              x2=x(i)
              y2=y(i)
              x3=x(i+1)
              y3=y(i+1)
            else
              x1=x(i)
              y1=y(i)
              x2=x(i+1)
              y2=y(i+1)
              x3=x(i+2)
              y3=y(i+2)
            endif
          else if (i == n) then
            if (xmirnodes(i)) then
              x1=x(i-1)
              y1=y(i-1)
              x2=x(i)
              y2=y(i)
              x3=-x(i-1)
              y3=y(i-1)
            else if (ymirnodes(i)) then
              x1=x(i-1)
              y1=y(i-1)
              x2=x(i)
              y2=y(i)
              x3=x(i-1)
              y3=-y(i-1)
            else
              x1=x(i-2)
              y1=y(i-2)
              x2=x(i-1)
              y2=y(i-1)
              x3=x(i)
              y3=y(i)
            endif
          else
            x1=x(i-1)
            y1=y(i-1)
            x2=x(i)
            y2=y(i)
            x3=x(i+1)
            y3=y(i+1)
          endif
          call circleradcurv(x1,y1,x2,y2,x3,y3,elementsonright,maxrad, &
                             xc,yc,radcurv(i),clockwise(i),concave(i),det)
        enddo

        return
        end

!=======================================================================================================

        subroutine radcurvcircclosed(n,x,y,radcurv,maxrad,clockwise, &
                                     concave,elementsonright)

! Compute the radius of curvature at the (x(i),y(i)) coordinates, using
! the fact that the boundary is a closed curve. The maximum radius of
! curvature to be returned is maxrad, which allows for the case where
! any given three points are colinear. This would produce an infinite
! radius of curvature. The radius of curvature is computed by fitting
! a circle through three points at a time while moving along the
! supplied boundary.

        implicit none

        integer n,clockwise(*),concave(*)
```

```
      real*8  x(*),y(*),radcurv(*),maxrad
      logical elementsonright

      integer i
      real*8  det,x1,y1,x2,y2,x3,y3,xc,yc
      logical straightline

      do i=1,n
        if (i == 1) then
          x1=x(n)
          y1=y(n)
          x2=x(i)
          y2=y(i)
          x3=x(i+1)
          y3=y(i+1)
        else if (i == n) then
          x1=x(n-1)
          y1=y(n-1)
          x2=x(n)
          y2=y(n)
          x3=x(1)
          y3=y(1)
        else
          x1=x(i-1)
          y1=y(i-1)
          x2=x(i)
          y2=y(i)
          x3=x(i+1)
          y3=y(i+1)
        endif
        call circleradcurv(x1,y1,x2,y2,x3,y3,elementsonright,maxrad, &
                           xc,yc,radcurv(i),clockwise(i),concave(i),det)
      enddo

      return
      end

!=============================================================================================================

      subroutine circleradcurv(x1,y1,x2,y2,x3,y3,materialonright,maxrad, &
                               xc,yc,radcurv,clockwise,concave,det)

! Compute the radius of curvature at point (x2,y2) using the three
! points (x1,y1), (y2,x2) and (x3,y3). The maximum radius of curvature
! to be returned is maxrad, which allows for the case where the three
! points are colinear. This condition would normally produce an
! infinite radius of curvature. The radius of curvature is computed by
! fitting a circle through the three points.

      implicit none

      integer clockwise,concave
      real*8  det,x1,y1,x2,y2,x3,y3,xc,yc,radcurv,maxrad
      logical materialonright

      logical straightline

      call circle(x1,y1,x2,y2,x3,y3,xc,yc,radcurv,det)

      if (det == 0.0d0 .or. radcurv > maxrad) then
        radcurv=maxrad
        straightline=.true.
      else
        straightline=.false.
      endif

      if (straightline) then
        clockwise=0
        concave=0
      else
        call rotationdir(xc,yc,x1,y1,x2,y2,clockwise)
        if (materialonright) then
          if (clockwise == -1) then
            concave=1
          else
            concave=-1
          endif
```

```
          else
            if (clockwise == -1) then
              concave=-1
            else
              concave=1
            endif
          endif
        endif

        return
        end

!=======================================================================================================

        subroutine checkxybounds(nbndnodes,x,y,chkxbnd,xbndlft,xbndrgt,chkybnd,ybndtop,ybndbot, &
                                 actnodes,xfixnodes,yfixnodes)

        implicit none

        integer nbndnodes
        real*8  x(*),y(*),xbndlft,xbndrgt,ybndtop,ybndbot
        logical chkxbnd,chkybnd,actnodes(*),xfixnodes(*),yfixnodes(*)

        integer i

        if (chkxbnd) then
          do i=1,nbndnodes
            if (actnodes(i).and..not.(xfixnodes(i).and.yfixnodes(i))) then
              x(i)=max(x(i),xbndlft)
              x(i)=min(x(i),xbndrgt)
            endif
          enddo
        endif

        if (chkybnd) then
          do i=1,nbndnodes
            if (actnodes(i).and..not.(xfixnodes(i).and.yfixnodes(i))) then
              y(i)=min(y(i),ybndtop)
              y(i)=max(y(i),ybndbot)
            endif
          enddo
        endif

        return
        end

!=======================================================================================================

      subroutine checkxypolygon(allpointsinside,xyinside,nbndnodes,nodenum,x,y,actnodes, &
                                npolygon,xbndpolygon,ybndpolygon,showmsg)

      implicit none

      logical allpointsinside,xyinside(nbndnodes)
      integer nbndnodes,nodenum(nbndnodes)
      real*8  x(nbndnodes),y(nbndnodes)
      logical actnodes(nbndnodes)
      integer npolygon
      real*8  xbndpolygon(npolygon),ybndpolygon(npolygon)
      logical showmsg

      integer i,l,m

      allpointsinside=.true.
      do i=1,nbndnodes
        xyinside(i)=.true.
        if (actnodes(i)) then
          call dlocpt(x(i),y(i),xbndpolygon,ybndpolygon,npolygon,l,m)
          if (l == -1) then
            if (showmsg) write(*,'(1x,a,i13)') '*** Boundary node outside polygon    = ',nodenum(i)
            xyinside(i)=.false.
            allpointsinside=.false.
          else if (l == 0) then
            if (showmsg) write(*,'(1x,a,i13)') '*** Boundary node on polygon         = ',nodenum(i)
            xyinside(i)=.false.
          endif
        endif
```

```
      enddo

      return
      end

!=============================================================================================

      subroutine makexypolygon(nbndnodes,xn,yn,xpypinside,xp,yp,npolygon,xbndpolygon,ybndpolygon, &
                               actnodes,xfixnodes,yfixnodes,wasconstrained)

      implicit none

      integer nbndnodes,npolygon
      real*8  xn(nbndnodes),yn(nbndnodes)
      logical xpypinside(nbndnodes)
      real*8  xp(nbndnodes),yp(nbndnodes)
      real*8  xbndpolygon(npolygon),ybndpolygon(npolygon)
      logical actnodes(nbndnodes),xfixnodes(nbndnodes),yfixnodes(nbndnodes)
      logical wasconstrained

      integer i,ln,lp,m,num,ierr
      logical outside

      wasconstrained=.false.

      ! Determine whether any points on the moving boundary lie outside the bounding polygon.

      outside=.false.
      do i=1,nbndnodes
        if (actnodes(i) .and. .not.(xfixnodes(i).and.yfixnodes(i)) .and. xpypinside(i)) then
          call dlocpt(xn(i),yn(i),xbndpolygon,ybndpolygon,npolygon,ln,m)
          if (ln == -1) outside=.true.
        endif
      enddo

      ! If required, constrain the points on the boundary so that they lie
      ! either on or within the bounding polygon.

      do while (outside)
        wasconstrained=.true.
        do i=1,nbndnodes
          call dlocpt(xn(i),yn(i),xbndpolygon,ybndpolygon,npolygon,ln,m)
          if (actnodes(i) .and. ln == -1 .and. xpypinside(i)) then
            if (xfixnodes(i) .and. yfixnodes(i)) then
              xn(i)=xp(i)
              yn(i)=yp(i)
            else if (xfixnodes(i)) then
              xn(i)=xp(i)
              yn(i)=(yp(i)+yn(i))/2.0d0
            else if (yfixnodes(i)) then
              xn(i)=(xp(i)+xn(i))/2.0d0
              yn(i)=yp(i)
            else
              xn(i)=(xp(i)+xn(i))/2.0d0
              yn(i)=(yp(i)+yn(i))/2.0d0
            endif
          endif
        enddo
        outside=.false.
        do i=1,nbndnodes
          if (actnodes(i) .and. .not.(xfixnodes(i).and.yfixnodes(i)) .and. xpypinside(i)) then
            call dlocpt(xn(i),yn(i),xbndpolygon,ybndpolygon,npolygon,ln,m)
            if (ln == -1) outside=.true.
          endif
        enddo
      enddo

      return
      end

!=============================================================================================

      subroutine makexypolygon3(nbndnodes,xn,yn,xp,yp,npolygon,xbndpolygon,ybndpolygon, &
                                actnodes,xfixnodes,yfixnodes,wasconstrained)

      implicit none
```

```
      integer nbndnodes,npolygon
      real*8  xn(nbndnodes),yn(nbndnodes),xp(nbndnodes),yp(nbndnodes)
      real*8  xbndpolygon(npolygon),ybndpolygon(npolygon)
      logical actnodes(nbndnodes),xfixnodes(nbndnodes),yfixnodes(nbndnodes)
      logical wasconstrained

      integer i,ln,lp,m,num,ierr

      wasconstrained=.false.

      do i=1,nbndnodes
        if (actnodes(i) .and. .not.(xfixnodes(i).and.yfixnodes(i))) then
          ln=-1
          do while (ln == -1)
            ! Determine whether the point on the moving boundary lies outside,
            ! inside or on the bounding polygon.
            call dlocpt(xn(i),yn(i),xbndpolygon,ybndpolygon,npolygon,ln,m)
            call dlocpt(xp(i),yp(i),xbndpolygon,ybndpolygon,npolygon,lp,m)
            write(*,'(1x,a,3i6,2f16.10,2f10.4)') 'Point i, ln, lp =',i,ln,lp,xn(i),yn(i),xp(i),yp(i)
            if (ln == -1) then
              if (lp == 0) then
                write(*,*) '*** Point ',i,' on initial boundary is on the bounding polygon.'
                xn(i)=xp(i)
                yn(i)=yp(i)
                ln=0
              else if (lp == 1) then
                if (xfixnodes(i).and.yfixnodes(i)) then
                  xn(i)=xp(i)
                  yn(i)=yp(i)
                else if (xfixnodes(i)) then
                  xn(i)=xp(i)
                  yn(i)=(yp(i)+yn(i))/2.0d0
                else if (yfixnodes(i)) then
                  xn(i)=(xp(i)+xn(i))/2.0d0
                  yn(i)=yp(i)
                else
                  xn(i)=(xp(i)+xn(i))/2.0d0
                  yn(i)=(yp(i)+yn(i))/2.0d0
                endif
                wasconstrained=.true.
              else if (lp == -1) then
                write(*,*) '*** Point ',i,' on initial boundary is outside the bounding polygon.'
                stop
              endif
            endif
          enddo
        endif
      enddo

      return
      end

!=====================================================================================================

      subroutine makexypolygon2(nbndnodes,xn,yn,xp,yp,npolygon,xbndpolygon,ybndpolygon, &
                                actnodes,xfixnodes,yfixnodes,wasconstrained)

      implicit none

      integer nbndnodes,npolygon
      real*8  xn(nbndnodes),yn(nbndnodes),xp(nbndnodes),yp(nbndnodes)
      real*8  xbndpolygon(npolygon),ybndpolygon(npolygon)
      logical actnodes(nbndnodes),xfixnodes(nbndnodes),yfixnodes(nbndnodes)
      logical wasconstrained

      integer i,ln,lp,m,num,ierr
      real*8  xinter(npolygon),yinter(npolygon)

      wasconstrained=.false.

      do i=1,nbndnodes
        if (actnodes(i).and..not.(xfixnodes(i).and.yfixnodes(i))) then
          ! Determine whether the point on the moving boundary lies outside,
          ! inside or on the bounding polygon.
          call dlocpt(xn(i),yn(i),xbndpolygon,ybndpolygon,npolygon,ln,m)
          call dlocpt(xp(i),yp(i),xbndpolygon,ybndpolygon,npolygon,lp,m)
          write(*,'(a,3i6,4f10.4)') 'Point i, ln, lp =',i,ln,lp,xn(i),yn(i),xp(i),yp(i)
```

```
          if (lp == -1) then
            write(*,*) '*** Point ',i,' on initial boundary is outside the bounding polygon.'
          else if (lp == 0) then
            write(*,*) '*** Point ',i,' on initial boundary is on the bounding polygon.'
          endif
          ! If point is outside the bounding polygon, determine the intersection
          ! point of the movement vector and the polygon.
          if (ln == -1) then
            call dpfindpolygon(xn(i),yn(i),xp(i),yp(i),xbndpolygon,ybndpolygon, &
                               npolygon,xinter,yinter,npolygon,num,ierr)
            if (num /= 1) then
              call dlocpt(xp(i),yp(i),xbndpolygon,ybndpolygon,npolygon,lp,m)
              if (lp == 0) then
                ! Allow for case where (xp(i),yp(i)) lies on the polygon, but has
                ! not been detected as an intersection point.
                xn(i)=xp(i)
                yn(i)=yp(i)
                write(*,*) '*** Info: xp on polygon, but not detected as an intersection point.'
              else
                write(*,*) '*** Error: Invalid number of intersections with polygon.'
                write(*,*) '*** ln =',ln,'  lp =',lp
                write(*,*) '*** Number of intersections =',num,'  ierr = ',ierr
                write(*,*) '*** i =',i
                write(*,*) '*** xn(i),yn(i)=',xn(i),yn(i)
                write(*,*) '*** xp(i),yp(i)=',xp(i),yp(i)
                stop
              endif
            else
              xn(i)=xinter(1)
              yn(i)=yinter(1)
              wasconstrained=.true.
            endif
          endif
        enddo

        return
        end

!=======================================================================================================

        subroutine includedangle(x1,y1,z1,x2,y2,z2,x3,y3,z3,a)

! Given three consecutive points in space, compute the included angle
! corresponding to the second point. The Law of Cosines is used to
! explicitly compute the angle. If the three points are colinear, an
! angle of 180 degrees is returned. If the first and third points are
! coincident, an angle of 0 degrees is returned.

        implicit none

        real*8 x1,y1,z1,x2,y2,z2,x3,y3,z3,a

        real*8 aa,bb,cc

        ! Compute length of each of the three sides of the triangle.

        aa=sqrt((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)
        bb=sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
        cc=sqrt((x2-x3)**2+(y2-y3)**2+(z2-z3)**2)

        ! Compute the included angle in degrees using the
        ! formula for the Law of Cosines.

        a=acosd((bb**2+cc**2-aa**2)/(2.0d0*bb*cc))

        return
        end

!=======================================================================================================

        subroutine dvecmaxact(n,x,active,vecmax)

        implicit none

        integer n
        real*8  x(*)
```

```
      logical active(*)
      real*8  vecmax

      integer i,is

      is=1
      do while (.not.active(is))
        is=is+1
      enddo
      vecmax=x(is)

      do i=is,n
        if (active(i)) vecmax=max(vecmax,x(i))
      enddo

      return
      end

!=====================================================================================================

      subroutine dvecminact(n,x,active,vecmin)

      implicit none

      integer n
      real*8  x(*)
      logical active(*)
      real*8  vecmin

      integer i,is

      is=1
      do while (.not.active(is))
        is=is+1
      enddo
      vecmin=x(is)

      do i=is,n
        if (active(i)) vecmin=min(vecmin,x(i))
      enddo

      return
      end

!=====================================================================================================

      subroutine dvecavgact(n,x,active,vecavg)

      implicit none

      integer n
      real*8  x(*)
      logical active(*)
      real*8  vecavg

      integer i
      integer nactive
      real*8  xsum

      xsum=0.0d0
      nactive=0
      do i=1,n
        if (active(i)) then
          xsum=xsum+x(i)
          nactive=nactive+1
        endif
      enddo

      vecavg=xsum/nactive

      return
      end

!=====================================================================================================

      subroutine dvecminactconc(n,x,active,concave,dvecmin)
```

```
      implicit none

      integer n
      real*8  x(*)
      logical active(*)
      integer concave(*)
      real*8  dvecmin

      logical init
      integer i

      init=.true.
      do i=1,n
        if (active(i).and.concave(i) == 1) then
          if (init) then
            dvecmin=x(i)
            init=.false.
          endif
          dvecmin=min(dvecmin,x(i))
        endif
      enddo

      return
      end

!=====================================================================================================

      subroutine stress3d(sn,tnt,sxx,syy,szz,txy,tyz,tzx,dcx,dcy,dcz)

! The values (sxx syy szz txy tyz tzx) are the six stress components
! on planes that are perpendicular to the x, y, z axes. sx, sy, and
! sz are the computed stress components on an arbitrary oblique plane.
! The direction cosines of the normal n to this oblique plane are
! dcx, dcy, and dcz, respectively.

      implicit none

      real*8 sn,tnt,sxx,syy,szz,txy,tyz,tzx,dcx,dcy,dcz

      real*8 sx,sy,sz,tyx,tzy,txz

      tyx=txy
      tzy=tyz
      txz=tzx

      sx=dcx*sxx+dcy*tyx+dcz*tzx
      sy=dcx*txy+dcy*syy+dcz*tzy
      sz=dcx*txz+dcy*tyz+dcz*szz

      sn=dcx*sx+dcy*sy+dcz*sz
      tnt=sqrt(sx**2+sy**2+sz**2-sn**2)

      return
      end

!=====================================================================================================

      subroutine s1s2th1th2(sxx,syy,sxy,s1,s2,theta1,theta2)

! This subprogram computes the principal stresses s1 and s2 oriented
! at angles theta1 and theta2 using the stresses sxx, syy, and sxy.
!
! s1 is the numerically largest principal stress and theta1 is its
! associated angle.
!
! s2 is the numerically smallest principal stress and theta2 is its
! associated angle.
!
! Note that, by the definition of the principal angles obtained from
! "Mechanics of Materials" by Higdon et al., one of the principal
! stress angles lies between +/-45 degrees (inclusive), and the other
! is 90 degrees greater.

      implicit none

      real*8 sxx,syy,sxy,s1,s2,theta1,theta2
```

```
      real*8 ps1,ps2,pth1,pth2

      real*8 datand,dcosd,dsind

      if (sxx-syy /= 0.0d0) then
        pth1 = datand(2.0d0*sxy/(sxx-syy))/2.0d0
      else
        if (sxy > 0.0d0) then
          pth1 = +45.0d0
        else if (sxy < 0.0d0) then
          pth1 = -45.0d0
        else
          pth1 = 0.0d0
        endif
      endif

      pth2 = pth1 + 90.0d0

      if (pth2 > 90.0d0) pth2 = pth2 - 180.0d0

      ps1 = (sxx+syy)/2.0d0 + (sxx-syy)/2.0d0*dcosd(2.0d0*pth1) + &
            sxy*dsind(2.0d0*pth1)

      ps2 = (sxx+syy)/2.0d0 + (sxx-syy)/2.0d0*dcosd(2.0d0*pth2) + &
            sxy*dsind(2.0d0*pth2)

      if (ps1 >= ps2) then
        s1 = ps1
        s2 = ps2
        theta1 = pth1
        theta2 = pth2
      else
        s1 = ps2
        s2 = ps1
        theta1 = pth2
        theta2 = pth1
      endif

      return
      end

!=====================================================================================================

      logical function within(x,xtest,tol,bounddef)

! This function returns true if the value x is within the range defined
! by the value of tol when x is compared to the reference value given
! in xtest.
!
! The bounddef character*2 variable can take on the values '[]', '()',
! '(]', and '[)'. The bounds defined using the [] characters mean that
! the values are included in the limits, while the () characters mean
! that the values are excluded from the limits.

      implicit none

      real*8    x,xtest,tol
      character bounddef*2

      real*8    xminval,xmaxval

      if (xtest > 0.0d0) then
        xmaxval=xtest*(1.0d0+tol)
        xminval=xtest*(1.0d0-tol)
      else if (xtest < 0.0d0) then
        xmaxval=xtest*(1.0d0-tol)
        xminval=xtest*(1.0d0+tol)
      else
        write(*,*) '*** ERROR: within called with xtest = 0.0.'
        stop
      endif

      if (xminval > xmaxval) then
        write(*,*) '*** ERROR: within called with invalid limits.'
        write(*,*) '           xminval = ',xminval
        write(*,*) '           xmaxval = ',xmaxval
        stop
```

```
      else if (bounddef == '[]') then
        if (xminval <= x .and. x <= xmaxval) then
          within=.true.
        else
          within=.false.
        endif
      else if (bounddef == '()') then
        if (xminval < x .and. x < xmaxval) then
          within=.true.
        else
          within=.false.
        endif
      else if (bounddef == '[)') then
        if (xminval <= x .and. x < xmaxval) then
          within=.true.
        else
          within=.false.
        endif
      else if (bounddef == '(]') then
        if (xminval < x .and. x <= xmaxval) then
          within=.true.
        else
          within=.false.
        endif
      else
        write(*,*) '*** ERROR: within called using ''',bounddef,'''','.'
        stop
      endif

      return
      end

!=====================================================================================================

      real*8 function biggestof2(x1,x2)

      implicit none

      real*8 x1,x2

      if (abs(x2) > abs(x1)) then
        biggestof2=x2
      else
        biggestof2=x1
      endif

      return
      end

!=====================================================================================================

      real*8 function biggestof3(x1,x2,x3)

      implicit none

      real*8 x1,x2,x3
      real*8 x

      x=x1

      if (abs(x2) > abs(x)) then
        x=x2
      endif

      if (abs(x3) > abs(x)) then
        x=x3
      endif

      biggestof3=x

      return
      end

!=====================================================================================================

      logical function inrange(x,xminval,xmaxval,bounddef)
```

```
! This function returns true if the value x is within the range defined
! by the lower limit xminval and the upper limit xmaxval.
!
! The bounddef character*2 variable can take on the values '[]', '()',
! '(]', and '[)'. The bounds defined using the [] characters mean that
! the values are included in the limits, while the () characters mean
! that the values are excluded from the limits.

      implicit none

      real      x,xminval,xmaxval
      character bounddef*2

      if (xminval.gt.xmaxval) then
        write(*,*) '*** Error: function inrange called with invalid limits.'
        write(*,*) '           xminval = ',xminval
        write(*,*) '           xmaxval = ',xmaxval
        stop
      else if (bounddef.eq.'[]') then
        if (xminval.le.x .and. x.le.xmaxval) then
          inrange = .true.
        else
          inrange = .false.
        endif
      else if (bounddef.eq.'()') then
        if (xminval.lt.x .and. x.lt.xmaxval) then
          inrange = .true.
        else
          inrange = .false.
        endif
      else if (bounddef.eq.'[)') then
        if (xminval.le.x .and. x.lt.xmaxval) then
          inrange = .true.
        else
          inrange = .false.
        endif
      else if (bounddef.eq.'(]') then
        if (xminval.lt.x .and. x.le.xmaxval) then
          inrange = .true.
        else
          inrange = .false.
        endif
      else
        write(*,*) '*** Error: inrange called using ''',bounddef,''''
        stop
      endif

      return
      end
```

A.4. Supporting subroutines contained in locpt.f90

The subroutines `dlocpt` and `dpfind` are double-precision versions of the two subroutines `locpt` and `pfind` that were originally published by Morris (1993).

```
!=======================================================================

      subroutine dlocpt(x0,y0,x,y,n,l,m)

! 22-10-2002 WW
!
! Double precision version of locpt with separate argument for eps.
!
! Given a polygonal line connecting the vertices (x(i),y(i))
! (i = 1,...,n) taken in this order. It is assumed that the
! polygonal path is a loop, where (x(n),y(n)) = (x(1),y(1))
! or there is an arc from (x(n),y(n)) to (x(1),y(1)).
!
! (x0,y0) is an arbitrary point and l and m are variables.
! l and m are assigned the following values ...
!
!    l = -1   if (x0,y0) is outside the polygonal path
!    l =  0   if (x0,y0) lies on the polygonal path
!    l =  1   if (x0,y0) is inside the polygonal path
```

```
!
! m = 0 if (x0,y0) is on or outside the path. If (x0,y0)
! is inside the path then m is the winding number of the
! path around the point (x0,y0).
!
! eps is a machine dependent constant. eps is the smallest
! number such that 1.0 + eps .gt. 1.0

      implicit none

      real*8  x(n), y(n), x0, y0
      integer n, l, m

      real*8  eps, pi, pi2, tol, u, v, theta, theta1, thetai, angle, sum
      integer n0, i

      ! eps is a machine dependent constant. eps is the smallest
      ! number such that 1.0 + eps .gt. 1.0

      eps = epsilon(x0)

      if (1.0d0+eps.eq.1.0d0) then
        write(*,*) '*** Error in dlocpt: 1.0d0+eps = 1.0d0, eps=',eps
        stop
      endif

      n0 = n
      if (x(1) .eq. x(n) .and. y(1) .eq. y(n)) n0 = n - 1
      pi = atan2(0.0d0, -1.0d0)
      pi2 = 2.0d0*pi
      tol = 4.0d0*eps*pi
      l = -1
      m = 0

      u = x(1) - x0
      v = y(1) - y0
      if (u .eq. 0.0 .and. v .eq. 0.0) go to 20
      if (n0 .lt. 2) return
      theta1 = atan2(v, u)

      sum = 0.0d0
      theta = theta1
      do 10 i = 2,n0
        u = x(i) - x0
        v = y(i) - y0
        if (u .eq. 0.0d0 .and. v .eq. 0.0d0) go to 20
        thetai = atan2(v, u)
        angle = abs(thetai - theta)
        if (abs(angle - pi) .lt. tol) go to 20
        if (angle .gt. pi) angle = angle - pi2
        if (theta .gt. thetai) angle = -angle
        sum = sum + angle
        theta = thetai
   10 continue

      angle = abs(theta1 - theta)
      if (abs(angle - pi) .lt. tol) go to 20
      if (angle .gt. pi) angle = angle - pi2
      if (theta .gt. theta1) angle = -angle
      sum = sum + angle

      ! sum = 2*pi*m where m is the winding number

      m = abs(sum)/pi2 + 0.2d0
      if (m .eq. 0) return
      l = 1
      if (sum .lt. 0.0d0) m = -m
      return

      ! (x0, y0) is on the boundary of the path

   20 l = 0
      return
      end

!=======================================================================
```

```
      subroutine dpfind(a, b, x, y, n, u, v, m, num, ierr)

! 24-10-2002 WW
!
! Double precision version of pfind.
!
! Calculate intersection points of a straight line and a polygonal path.

      implicit none

      integer n,num,ierr
      real*8  a(2),b(2),x(n),y(n),u(m),v(m)

      real*8  hi,k,ki,eps,h,tol,tol0,onep,onem,d,p,q,s,t
      real*8  diff1,diff,tmin,tmax
      integer nm1,m,ind,i

      ! eps is a machine dependent constant. eps is the
      ! smallest number such that 1.0 + eps .gt. 1.0 .

      eps = epsilon(h)

      num = 0
      if (n .lt. 2) go to 200
      h = b(1) - a(1)
      k = b(2) - a(2)
      if (h .eq. 0.0d0 .and. k .eq. 0.0d0) go to 200

      ierr = 0
      nm1 = n - 1
      tol = 4.0*eps
      tol0 = 2.0*eps
      onep = 1.0 + tol
      onem = 0.5 + (0.5 - tol0)

      ind = 0
      do 100 i = 1, nm1
         hi = x(i + 1) - x(i)
         ki = y(i + 1) - y(i)
         if (hi .eq. 0.0d0 .and. ki .eq. 0.0d0) go to 100
         ind = 1

         ! Check if the line from a to b and the i-th line in the
         ! path are parallel.

         s = hi*k
         t = h*ki
         d = s - t
         if (abs(d) .le. tol*max(abs(s),abs(t))) go to 40

         ! The lines are not parallel.

         p = x(i) - a(1)
         q = y(i) - a(2)
         s = hi*q
         t = ki*p
         diff = s - t
         if (abs(diff) .le. tol*max(abs(s),abs(t))) diff = 0.0d0
         s = h*q
         t = k*p
         diff1 = s - t
         if (abs(diff1) .le. tol*max(abs(s),abs(t))) diff1 = 0.0d0

         s = diff/d
         t = diff1/d
         if (s .lt. 0.0d0 .or. s .gt. onep) go to 100
         if (t .lt. 0.0d0 .or. t .gt. onep) go to 100
         if (num .gt. 0 .and. t .eq. 0.0d0) go to 100
         if (s .gt. 0.0d0) go to 20

         ! Point a is on the i-th line.

   10    num = num + 1
         if (num .gt. m) go to 210
         u(num) = a(1)
         v(num) = a(2)
         go to 100
```

```
      ! Point b is on the i-th line

 20   if (s .lt. onem) go to 30
 21   num = num + 1
      if (num .gt. m) go to 210
      u(num) = b(1)
      v(num) = b(2)
      go to 100

      ! The interior of the line from a to b intersects with the i-th line.

 30   num = num + 1
      if (num .gt. m) go to 210
      u(num) = a(1) + s*h
      v(num) = a(2) + s*k
      go to 100

      ! The lines are parallel.

 40   if (abs(hi) .gt. abs(ki)) go to 50

         d = a(2) - y(i)
         if (abs(d) .le. tol0*max(abs(a(2)),abs(y(i)))) d = 0.0d0
         s = d/ki

         p = x(i) + s*hi
         if (abs(a(1) - p) .gt. tol*max(abs(a(1)),abs(p))) go to 100

         d = b(2) - y(i)
         if (abs(d) .le. tol0*max(abs(b(2)),abs(y(i)))) d = 0.0d0
         t = d/ki
         go to 60

 50   d = a(1) - x(i)
      if (abs(d) .le. tol0*max(abs(a(1)),abs(x(i)))) d = 0.0d0
      s = d/hi

      p = y(i) + s*ki
      if (abs(p - a(2)) .gt. tol*max(abs(p),abs(a(2)))) go to 100

      d = b(1) - x(i)
      if (abs(d) .le. tol0*max(abs(b(1)),abs(x(i)))) d = 0.0d0
      t = d/hi

      ! The 2 lines are portions of the same straight infinite line.

 60   if (s .gt. 0.0d0 .and. s .lt. onem) go to 220
      if (t .gt. 0.0d0 .and. t .lt. onem) go to 220
      tmin = min(s,t)
      tmax = max(s,t)
      if (tmax .le. 0.0d0) go to 70
      if (tmin .ge. onem) go to 80
      go to 220

 70   if (tmax .lt. 0.0d0) go to 100
      if (num .gt. 0) go to 100
      if (tmax .eq. s) go to 10
      go to 21

 80   if (tmin .gt. 1.0) go to 100
      if (tmin .eq. s) go to 10
      go to 21

100 continue
    if (ind .eq. 0) go to 200

    if (num .lt. 2) return
    if (u(num) .eq. x(1) .and. v(num) .eq. y(1)) num = num - 1
    return

    ! Error return.

200 ierr = 1
    return
210 ierr = 2
    num = num - 1
```

```
      return
  220 ierr = -i
      return
      end

!=====================================================================

      subroutine dpfindpolygon(xa,ya,xb,yb,x,y,n,u,v,m,num,ierr)

! Calculate intersection of a straight line and a polygon (as distinct
! from a polygonal path).

      implicit none

      integer n,m,num,m,ierr
      real*8  x(n),y(n),u(m),v(m),xa,ya,xb,yb

      integer np,i
      real*8  xp(n+1),yp(n+1),a(2),b(2)

      np=n+1

      do i=1,n
        xp(i)=x(i)
        yp(i)=y(i)
      enddo
      xp(np)=x(1)
      yp(np)=y(1)

      a(1)=xa
      a(2)=ya
      b(1)=xb
      b(2)=yb

      call dpfind(a,b,xp,yp,np,u,v,m,num,ierr)

      return
      end
```

A.5. Supporting functions and subroutines contained in arczerocurv.f90

```
!=====================================================================

      subroutine getzerocross(npts,f,nzc,izcbeg,izcend,nzones,izone)

! Determine the number of zero crossings, nzc, in the function stored
! in the vector f(npts). Also determine the number of zones, nzones,
! where the function has a positive or negative sign in between any
! zero crossings, which may or may not be present.
!
! The start and end of the subinterval containing the zero crossing is
! returned in the vectors izcbeg and izcend, respectively.
!
! For each point i, the array izone(i) stores the identification number
! of the interval in which point i lies between any two zero crossings.
!
! If no zero crossings are present, then nzc=0 and izone(i)=1 for i=1
! to i=npts.
!
! Zero crossings are identified by a change in sign of the value of the
! function at each point when compared to the sign of the sum of the
! function values in the preceding zone. In this way, a function that
! has zero values, but is otherwise positive or negative signed, does
! not have any zero "crossings".

      implicit none

      integer npts,nzc,nzones
      integer izcbeg(*),izcend(*),izone(*)
      real*8  f(npts)

      integer i,j
      real*8  sum

      nzc=0
      izone(1)=1
```

```
      sum=0.0d0

      do i=1,npts-1
        j=i+1
        sum=sum+f(i)
        if (sum*f(j).lt.0.0d0) then
          nzc=nzc+1
          izcbeg(nzc)=i
          izcend(nzc)=j
          sum=0.0d0
        endif
        izone(j)=nzc+1
      enddo

      nzones=nzc+1

      return
      end

!=======================================================================

      subroutine getzerocrossclosed(npts,f,nzc,izcbeg,izcend,nzones, &
                                    izone)

! Determine the number of zero crossings, nzc, in the function stored
! in the vector f(npts). Also determine the number of zones, nzones,
! where the function has a positive or negative sign in between any
! zero crossings, which may or may not be present.
!
! The function f is assumed to correspond to to a closed curve, where
! the point at f(npts) is followed by the point at f(1).
!
! The start and end of the subinterval containing the zero crossing is
! returned in the vectors izcbeg and izcend, respectively.
!
! For each point i, the array izone(i) stores the identification number
! of the interval in which point i lies between any two zero crossings.
!
! If no zero crossings are present, then nzc=0 and izone(i)=1 for i=1
! to i=npts.
!
! Zero crossings are identified by a change in sign of the value of the
! function at each point when compared to the sign of the sum of the
! function values in the preceding zone. In this way, a function that
! has zero values, but is otherwise positive or negative signed, does
! not have any zero "crossings".

      implicit none

      integer npts,nzc,nzones
      integer izcbeg(*),izcend(*),izone(*)
      real*8  f(npts)

      real*8  sum,favg(npts),fpeak(npts)
      integer i,j,ipeak(npts)
      integer mod

      if (npts.lt.2) then
        write(*,*)
        write(*,*) 'npts=',npts,'(<2) in subroutine getzerocrossclosed.'
        stop
      endif

      nzc=0
      izone(1)=1
      sum=0.0d0

      do i=1,npts-1
        j=i+1
        sum=sum+f(i)
        if (sum*f(j).lt.0.0d0) then
          nzc=nzc+1
          izcbeg(nzc)=i
          izcend(nzc)=j
          sum=0.0d0
        endif
        izone(j)=nzc+1
```

```
      enddo

      nzones=nzc+1

      ! Check for a sign change across the end zone and the starting zone.

      if (nzc.gt.0) then
        call getpkavgbwzone(npts,f,nzones,izone,fpeak,favg,ipeak)
        if (favg(nzones)*favg(1).gt.0.0d0) then
          do i=1,npts
            if (izone(i).eq.nzones) izone(i)=1
          enddo
          nzones=nzones-1
        endif
      endif

      return
      end

!=======================================================================

      subroutine getpkavgbwzone(npts,f,nzones,izone,fpeak,favg,ipeak)

! Get the peak and average values of the function f in each zone.
! Each zone is assumed to contain either positive values or negative
! values only, and may include a point with a value of zero.

      implicit none

      integer npts,nzones
      integer izone(npts),ipeak(nzones)
      real*8  f(npts),fpeak(nzones),favg(nzones)

      real*8  sum
      integer i,j,n

      ! Find the peak and average values in each zone.

      do i=1,nzones
        do j=1,npts
          if (izone(j).eq.i) then
            ipeak(i)=j
            fpeak(i)=f(j)
            exit
          endif
        enddo
        sum=0.0d0
        n=0
        do j=1,npts
          if (izone(j).eq.i) then
            n=n+1
            sum=sum+f(j)
            if (abs(f(j)).gt.abs(fpeak(i))) then
              ipeak(i)=j
              fpeak(i)=f(j)
            endif
          endif
        enddo
        favg(i)=sum/n
      enddo

      return
      end

!=======================================================================

      subroutine getstatszone(npts,f,nzones,izone,ftol,fpeak,fmin, &
                              favg,fstddev,frange,ipeak)

! Get the peak, minimum, average, standard deviation, and range
! values of the function f in each zone. The statistics are computed
! for function values that lie within the range defined by
! abs((1.0-ftol)*fpeak) to abs(fpeak), inclusive. In this case, ftol
! is a relative tolerance. The code assumes that the zone consists of
! values that are all positive (with a possible zero value) or all
! negative (with a possible zero value).
```

```
      implicit none

      integer npts,nzones
      integer izone(npts),ipeak(nzones)
      real*8  f(npts),favg(nzones),fstddev(nzones)
      real*8  frange(nzones),fpeak(nzones),fmin(nzones)
      real*8  ftol

      real*8  sumf,sumff,variancef
      integer iz,ip,n

      ! Proceed to find the peak, average, standard deviation and
      ! range values for each individual zone.

      do iz=1,nzones

        ! Initialize the process for finding the peak.

        do ip=1,npts
          if (izone(ip).eq.iz) then
            ipeak(iz)=ip
            fpeak(iz)=f(ip)
            exit
          endif
        enddo

        ! Find the peak (positive or negative) value within the
        ! zone, and let the minimum value start off being equal
        ! to this peak value.

        do ip=1,npts
          if (izone(ip).eq.iz) then
            if (abs(f(ip)).gt.abs(fpeak(iz))) then
              ipeak(iz)=ip
              fpeak(iz)=f(ip)
            endif
          endif
        enddo
        fmin(iz)=fpeak(iz)

        ! Determine sums of values and other statistics.

        sumf=0.0d0
        sumff=0.0d0
        n=0
        do ip=1,npts
          if (izone(ip).eq.iz) then
            if (abs(f(ip)).ge.abs((1.0d0-ftol)*fpeak(iz))) then
              n=n+1
              sumf=sumf+f(ip)
              sumff=sumff+f(ip)**2
              if (abs(f(ip)).lt.abs(fmin(iz))) fmin(iz)=f(ip)
            endif
          endif
        enddo

        if (n.gt.1) then
          variancef=(sumff - sumf*sumf/dble(n))/dble(n-1)
        else
          variancef=0.0d0
        endif

        fstddev(iz)=sqrt(abs(variancef))
        favg(iz)=sumf/n
        frange(iz)=abs(fpeak(iz)-fmin(iz))

      enddo

      return
      end

!=======================================================================

      real*8 function curvat(xt,yt,xtt,ytt)

! This function computes the curvature of a curve that is given in
! parametric form, x=f(t) and y=g(t). The input values xt and yt are
```

```
! the first derivatives with respect to t, and xtt and ytt are the
! second derivatives with respect to t. On a straight line the curvature
! will of course be zero.
!
! The formula used here was obtained from Eq. 2(c) in "Calculus and
! Analytic Geometry", 4th edition, by George B. Thomas, Jr., 1977.

      implicit none

      real*8 xt,yt,xtt,ytt

      curvat=abs(xt*ytt-yt*xtt)/(xt**2+yt**2)**1.5d0

      return
      end

!=======================================================================

      subroutine polygarclen(n,x,y,s,sn)

! Compute the polygonal arc length s and its normalised representation
! sn, using the assumption that the points represent a non-closed curve.
! The total arc length is found in s(n), and corresponds to a
! normalised arc length of sn(n)=1.0.

      implicit none

      integer n
      real*8  x(n),y(n),s(n),sn(n)

      integer i

      ! Determine polygonal arc length.

      s(1) = 0.0d0
      do i = 2,n
        s(i) = s(i-1)+sqrt((x(i)-x(i-1))**2+(y(i)-y(i-1))**2)
      enddo

      ! Normalise polygonal arc length.

      do i=1,n
        sn(i)=s(i)/s(n)
      enddo

      return
      end

!=======================================================================

      subroutine polygarclenp(n,x,y,np1,s,sn)

! Compute the polygonal arc length s and its normalised representation
! sn, using the assumption that the points represent a closed curve.
! The total arc length is returned in s(n+1), and corresponds to a
! normalised polygonal arc length of sn(n+1)=1.0.

      implicit none

      integer n,np1
      real*8  x(n),y(n),s(np1),sn(np1)

      integer i

      ! Determine the polygonal arc length.

      s(1) = 0.0e0
      do i = 2,n
        s(i) = s(i-1)+sqrt((x(i)-x(i-1))**2+(y(i)-y(i-1))**2)
      enddo
      s(n+1) = s(n)+sqrt((x(n)-x(1))**2+(y(n)-y(1))**2)

      ! Normalise the polygonal arc length.

      do i=1,np1
        sn(i)=s(i)/s(np1)
      enddo
```

```
      return
      end

!=======================================================================

      subroutine polygarclenc(n,x,y,s,sn,stotal)

! Compute the polygonal arc length s and its normalised representation
! sn, using the assumption that the points represent a closed curve.
! The total arc length is returned in stotal, and corresponds to a
! normalised polygonal arc length of 1.0. It includes the implied side
! of the polygon between the last and first point in the list.

      implicit none

      integer n
      real*8  x(n),y(n),s(n),sn(n),stotal

      integer i

      ! Determine the polygonal arc length.

      s(1) = 0.0d0
      do i = 2,n
        s(i) = s(i-1)+sqrt((x(i)-x(i-1))**2+(y(i)-y(i-1))**2)
      enddo
      stotal = s(n)+sqrt((x(n)-x(1))**2+(y(n)-y(1))**2)

      ! Normalise the polygonal arc length.

      do i=1,n
        sn(i)=s(i)/stotal
      enddo

      return
      end
```

A.6. Supporting routines contained in fparser.f90 and parameters.f90

The subroutines and functions contained in `fparser.f90` comprise a public domain function parser module fparser written in FORTRAN 90. The source code for fparser is available from the following location:

http://www.its.uni-karlsruhe.de/~schmehl/opensource/fparser-v1.0.tar.gz

Although some of the fparser routines are referenced in `optpafec042.f90`, this feature should not be used as it was never fully implemented in the shape optimisation program. They are only needed in the present version in order to get the shape optimisation program to compile correctly.

A.7. Supporting routines contained in dfitpack.f90

The subroutines and functions contained in `dfitpack.f90` come from a public domain software package called FITPACK, which is used for general-purpose curve and surface fitting. The source code for FITPACK was obtained from the *Netlib Repository* (http://www.netlib.org/), which is a collection of mathematical software, papers, and databases of software. The FITPACK software is available from the following location:

http://www.netlib.org/fitpack/all

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. DLM/CAVEAT (OF DOCUMENT)

2. TITLE	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)
Shape Optimisation of Holes in Loaded Plates by Minimisation of Multiple Stress Peaks	Document (U) Title (U) Abstract (U)

4. AUTHOR(S)	5. CORPORATE AUTHOR
Witold Waldman and Manfred Heller	DSTO Defence Science and Technology Organisation 506 Lorimer St Fishermans Bend Victoria 3207 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-RR-0412	AR-016-270	Research Report	April 2015

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2015/1016721	AIR 07/283	OIC-ASI-DGTA	123	35

13. DSTO PUBLICATIONS REPOSITORY	14. RELEASE AUTHORITY
http://dspace.dsto.defence.gov.au/dspace/	Chief, Aerospace Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

Yes

18. DSTO RESEARCH LIBRARY THESAURUS

Shape optimisation, Holes, Aircraft structure, Stress concentration, Fatigue life extension, Finite element method, Numerical modelling, Numerical simulation

19. ABSTRACT

A new method has been developed for simultaneously minimising the peak tangential stresses on multiple segments around the boundary of a hole in a uniaxially-loaded or biaxially-loaded plate. It is based upon iterative finite element analysis. The efficacy of this new multi-peak method is demonstrated by 2D and 3D numerical examples, some of which include significant geometric constraints. A comprehensive series of benchmarks is explored in some detail and sets of useful transferable coordinates of optimised hole shapes for selected load and geometry cases are provided. It is shown that, when the optimal shape is achieved, the separate tensile and compressive stress segments around the hole boundary converge to constant stress regions of different values. The optimal hole shapes produce significant reductions in peak stress for all regions around the hole boundary, as compared to typical non-optimal circular holes. As the most fatigue critical location in a structure may not necessarily be the one with the biggest active tensile peak, it is desirable to be able to minimise these other stress peaks around the hole boundary. Hence, optimal shapes computed using the multi-peak method are useful for critical structural regions where it is desirable to substantially increase or maximise the fatigue life.